EXHIBIT V

EXHIBIT 12
UNREDACTED VERSION OF DOCUMENT SOUGHT TO BE SEALED

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN JOSE DIVISION

CISCO SYSTEMS, INC., Plaintiff, v. ARISTA NETWORKS, INC., Defendant.	Case No. 5:14-cv-05344-BLF (PSG) **OPENING EXPERT REPORT OF DOUGLAS W. CLARK REGARDING INVALIDITY OF U.S. PATENT NO. 7,047,526**

CONTAINS MATERIAL DESIGNATED HIGHLY CONFIDENTIAL—ATTORNEYS' EYES ONLY

Executed on the 3 of June 2016 in Philadelphia, PA.

[illegible]

Douglas W. Clark

TABLE OF CONTENTS

Page

I. INTRODUCTION

1. My name is Douglas W. Clark and I have been retained by counsel for Defendant Arista Networks Inc. ("Arista") in this case to analyze and opine on the validity of U.S. Patent No. 7,047,526 ("the '526 patent"), asserted in this matter by Plaintiff Cisco Systems, Inc. ("Cisco").

2. I am being compensated for my work in this matter at my standard rate of $700 per hour and am being reimbursed for any expenses incurred in relation to my work on this matter. My compensation is not contingent upon the outcome of this matter or the substance of my testimony.

3. I expect to be called to provide expert testimony regarding opinions formed as a result of my analysis of the issues considered in this report if asked to do so by the Court or counsel for Arista. If asked to do so, I may also provide testimony describing the state of the art before and during the time of the alleged invention, including the characteristics and operation of parse trees, computer command line parsers, and related technology.

4. In reaching the conclusions described herein, I have considered the documents and materials identified in Exhibit A. My opinions are also based upon my education, training, research, knowledge, and personal and professional experience.

5. I reserve the right to modify or supplement my opinions, as well as the basis for my opinions, in light of any documents, testimony, expert opinions, or other evidence or information that may emerge during the course of this litigation, including any depositions that have yet to be taken.

6. I also understand that Cisco may submit an expert report responding to this report. I reserve the right to rebut any opinions offered in that report or any other issues asserted by Cisco.

II. QUALIFICATIONS AND EXPERIENCE

7. I received a Ph.D. in computer science from Carnegie-Mellon University in 1976 and a Bachelor of Science in engineering and applied science from Yale University in 1972. Since receiving my doctorate, I have devoted my professional career to the research, design, development, study, and teaching of numerous aspects of computer systems architecture and design. I have studied, taught, practiced, and researched in the field of computer science for over forty years.

8. I am currently Professor Emeritus of Computer Science at Princeton University, having recently retired. I was a Professor of Computer Science at Princeton from 1993 through January 2016. I taught numerous courses at Princeton, including an introductory computer science course, a foundational course for first-year engineering students, and advanced courses in computer architecture for upper-level undergraduates and graduate students. I also taught a number of advanced graduate seminars about various computer science topics, including the great papers of the field. I taught most of these courses several times. In addition, in 1990-91, I taught as a Visiting Lecturer in the Division of Applied Sciences at Harvard, and in 2003, I was a Visiting Professor of Computing and Information Science at the University of Pennsylvania.

9. In addition to my experience in academe, I have seventeen years of industrial experience designing computer systems. From 1976 to 1980, I was a Member of the Research Staff at the Xerox Palo Alto Research Center, where I worked chiefly on the design of the Dorado, one of the earliest high-performance workstations. From 1980 to 1993, I worked for the

Digital Equipment Corp., first as a Principal Engineer in the Systems Architecture Group, and then as a Consulting and Senior Consulting Engineer in both the Advanced VAX Systems Engineering and Alpha Advanced Development groups. I worked mainly on the architecture, organization, design, simulation, and performance analysis of VAX and Alpha computers. I was one of the principal designers of the VAX 8700 and VAX 8800—both highly successful machines of the late 1980's.

10. I have authored or co-authored about 60 academic publications in the fields of computer science and engineering. In addition, I have been a referee or associate editor for the following academic journals: ACM Transactions on Computers, IEEE Transactions on Computers, and IEEE Computer. My curriculum vitae includes a list of selected publications on which I am a named author.

11. I have also been a program committee member or co-chair at numerous national and international conferences/symposiums, including the International Conference on Computer Design, SIGMETRICS Conference on Measurement and Modeling of Computer Systems, and International Symposium on Computer Architecture.

12. A copy of my curriculum vitae is attached as Exhibit B, which contains further details regarding my experience, education, publications, and my other qualifications to render an expert opinion in connection with this matter.

III. MATERIALS RELIED UPON

13. As part of my preparation for writing this report, I have reviewed and considered the materials identified in Exhibit A to this report.

14. Below I provide the details of my analysis, and the conclusions that form the basis of any testimony that I give below. Such testimony I may include appropriate visual aids, some

or all of the data or other documents and information cited herein or identified in Exhibit A, and additional data or other information identified in discovery, to support or summarize my opinions. For example, I may include information cited by Cisco's expert(s), as well as witness testimony.

IV. LEVEL OF ORDINARY SKILL

15. I understand that a person having ordinary skill in the art is a hypothetical person who is used to analyze the prior art without the benefit of hindsight. I further understand that a person of ordinary skill in the art is presumed to be one who thinks along the lines of conventional wisdom in the art and is not one who undertakes to innovate, whether by extraordinary insights or by patient and often expensive systematic research.

16. I understand that the hypothetical person of ordinary skill is presumed to have knowledge of all references that are sufficiently related to one another and to the pertinent art, and to have knowledge of all arts reasonably pertinent to the particular problem that the claimed invention addresses.

17. In my opinion, a person of ordinary skill in the art pertaining to the patent at issue at the time of its filing would have at least a bachelor's degree in computer science and 3-5 years of experience in systems development.

V. LEGAL STANDARDS

18. I have been informed by counsel for Arista of the relevant legal standards that apply in evaluating the validity of a patent. I am not an attorney and am relying only on instructions from Arista's attorneys for these legal standards. Below I describe my understanding of these legal standards.

A. Burden of proof regarding patent invalidity

19. I understand that patents are presumed valid and that invalidity must be shown by "clear and convincing" evidence. I understand that the "clear and convincing" standard is higher than a preponderance of the evidence but lower than beyond a reasonable doubt, and that clear and convincing evidence is evidence that produces an abiding conviction that the truth of a fact is highly probable.

B. Priority date and scope of prior art

20. I have been instructed to treat the earliest filing date of the '526 patent—June 28, 2000—as the relevant priority date. I have also been instructed to use this date as the "critical date" upon which the information that I rely on herein qualifies as prior art.

21. The prior art that I rely on herein includes patents and other printed publications as well as knowledge, public use, and sale of any system or device that predate the critical date of the '526 patent. I reserve the right to rely on other, additional prior art should the priority or critical date for the '526 patent be deemed different from the one above.

22. I understand that Cisco contends that the priority date of the '526 patent is November 17, 1999, on the basis that the claimed inventions of the '526 patent were conceived and reduced to practice by that date. I further understand that it is Cisco's burden to establish a priority date that predates the filing of the patent application for the'526 patent. I reserve the right to supplement or amend my opinions in response to argument from Cisco that it had in fact conceived and reduced the claimed inventions of the '526 patent to practice prior to the filing date of June 28, 2000.

C. Claim Construction

23. I have been instructed and understand that claim construction is a matter of law for the Court to decide. I understand that the Court has not yet issued an order construing the terms of the '526 patent. Unless otherwise stated, I have given claim terms their ordinary and customary meaning within the context of the patent in which the terms are used, *i.e.*, the meaning that the term would have to a person of ordinary skill in the art in question at the time of the invention in light of what the patent teaches.

24. In addition, I have read and considered the proposed claim constructions in the following chart, which is derived from the Amended Joint Claim Construction Chart filed by the parties on March 5, 2016 (Exhibit C):

Term	**Cisco's Proposed Construction**	**Arista's Proposed Construction**
"management programs"	*Proposed construction:* "separate tools or external agents having their own respective command formats that provide management functions"	*Proposed construction:* "tools that are configured to execute user-entered commands having their own respective command formats rather than the generic command format" This proposal assumes that the term "generic command format" is not found to be indefinite.
"generic command"	*Proposed construction:* "command that provides an abstraction of the tool-specific command formats and syntax, enabling a user to issue the command based on the relative functions, as opposed to the specific syntax for a corresponding tool"	*Proposed construction:* Indefinite, OR if not indefinite: "command having a format and syntax that is an abstraction of the command formats and syntaxes of more than one management program, as opposed to the specific syntax for any such management program"

Term	**Cisco's Proposed Construction**	**Arista's Proposed Construction**
"command parse tree"	*Proposed construction:* "a hierarchical data representation having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value"	*Proposed construction:* "tree": "data structure consisting of linked nodes, with a root node (a node with no parent nodes), and where the remaining nodes are either a branch node (a node with a parent node and one or more children nodes), or a leaf node (a node with a parent node and no children nodes)" "command parse tree": "tree for interpreting commands where each node, or element, corresponds to one or more command components"
"the validating step including identifying one of the elements as a best match relative to the generic command"	*Proposed construction:* Plain and ordinary meaning (except that specific terms appearing within the phrase should be construed as proposed above)	*Proposed construction:* Indefinite, OR if not indefinite: "the validating step having the capability of both identifying the element in the parse tree that exactly matches the generic command, and, in the absence of an exact match, identifying the element that contains the last validated component of the generic command"
"the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value"	*Proposed construction:* Plain and ordinary meaning (except that specific terms appearing within the phrase should be construed as proposed above)	*Proposed construction:* "elements": "nodes" "command action value": "piece of data that uniquely represents the prescribed command." the entire phrase: "the command parse tree having nodes, such that each node specifies a unique command action value for each generic command component."

Term	**Cisco's Proposed Construction**	**Arista's Proposed Construction**
"means for validating a generic command received from a user, the validating means configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating means identifying one of the elements as a best match relative to the generic command"	*Proposed construction:* Function: validating a generic command received from a user Structure: Parser 14 in Figure 2, which includes the command word translation table 20 and the command parse tree 22, as described in 3:36- 61, and equivalents	*Proposed construction:* Functions: (1) validating a generic command received from a user (2) specifying valid generic commands relative to a prescribed generic command format, (3) having elements each specifying at least one corresponding generic component and a corresponding at least one command action value, and (4) identifying one of the elements as a best match relative to the generic command. Disclosed structure: A processor executing a parser, and a corresponding memory storing a command parse tree, wherein the parser executes the algorithm of Figure 3, and wherein (1) each node of the command parse tree specifies one token and a corresponding command key; (2) the top-level nodes of the command parse tree represent all possible valid first words in the input command, second-level nodes represent all possible valid second words for each valid first word in the input command, and so on.

25. I have also read the briefing and declarations submitted in support of those proposed constructions. Where the adoption of one party's proposed construction might impact the invalidity analysis of a prior art reference, I have noted that construction in my analysis.

26. I reserve the right to supplement or amend my opinions, to the extent warranted, once a claim construction order is issued by the Court.

D. Anticipation

27. I understand that a patent claim is "anticipated" if each and every element of the claim, as properly construed, has been disclosed in a single prior art reference either expressly or inherently (i.e., necessarily present even if not expressly stated), and the claimed arrangement or combination of those elements has been disclosed, either expressly or inherently, in the same prior art reference. I understand that prior art includes various categories of information such as printed publications, patents, actual commercial products, and/or other physical embodiments.

28. I understand that although anticipation cannot be established by combining references, additional references may be used to interpret the anticipating reference by, for example, indicating what the anticipating reference would have meant to one having ordinary skill in the art. In addition, the description provided in the prior art must be such that a person of ordinary skill in the art in the field of invention could, based on the reference, practice the invention without undue experimentation at the time of the alleged invention of the asserted patent.

E. Obviousness

29. I understand that a patent claim is rendered obvious if the claimed subject matter as a whole would have been obvious to a person of ordinary skill in the art as of the date of invention. I understand that this determination is made after weighing the following factors: (1)

level of ordinary skill in the pertinent art; (2) the scope and content of the prior art; (3) the differences between the prior art as a whole and the claim at issue; and (4) as appropriate, secondary considerations of non-obviousness.

30. I understand that the knowledge and understanding of a person of ordinary skill in the art at the time of the alleged invention provides a reference point from which the prior art and claimed invention should be viewed. This reference prevents such persons from using their own insight or hindsight in deciding whether a claim is obvious. However, if a person of ordinary skill in the art could have implemented the claimed invention as a predictable variation of a known product, then the claim may be rendered obvious. I further understand that a person of ordinary skill in the art is presumed to have knowledge of the relevant prior art at the time of the claimed invention, and that the scope of the prior art comprises any prior art that was reasonably pertinent to the particular problems the inventor faced.

31. I understand that an obviousness evaluation can be made on a single reference or a combination of multiple prior art references. I understand that a proper obviousness analysis as to two or more references generally requires a reason that would have prompted a person of ordinary skill in the relevant field to combine the elements of multiple prior art references in the way the claimed new invention does. I understand that the prior art references themselves may provide a suggestion, motivation, or reason to combine, but that at other times the nexus linking two or more prior art references is simple common sense. I further understand that obviousness analysis recognizes that market demand, rather than scientific literature, often drives innovation, and that a motivation to combine references may be supplied by the direction of the marketplace.

32. I understand that a particular combination may be proven obvious merely by showing that it was obvious to try the combination. For example, when there is a design need or

market pressure to solve a problem and there are a finite number of identified, predictable solutions, a person of ordinary skill has good reason to pursue the known options within that person's technical grasp because the result is likely the product not of innovation but of ordinary skill and common sense.

33. I further understand that a proper obviousness analysis focuses on what was known or obvious to a person of ordinary skill in the art, and not just the patentee. Accordingly, I understand that any need or problem known in the field of endeavor at the time of invention and addressed by the patent can provide a reason for combining the elements in the manner claimed.

34. In sum, my understanding is that prior art teachings are properly combined where a person of ordinary skill in the art, having the understanding and knowledge reflected in the prior art and motivated by the general problem facing the inventor, would have been led to make the combination of elements recited in the claims. Under this analysis, the prior art references themselves, or any need or problem known in the field of endeavor at the time of the invention, can provide a reason for combining the elements of multiple prior art references in the claimed manner.

35. I understand that at least the following rationales may support a finding of obviousness:

i. Combining prior art elements according to known methods to yield predictable results;

ii. Simple substitution of one known element for another to obtain predictable results;

iii. Use of a known technique to improve similar devices (methods, or products) in the same way;

iv. Applying a known technique to a known device (method, or product) ready for improvement to yield predictable results;

v. Obvious to try—choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success;

vi. A predictable variation of work in the same or a different field of endeavor if a person of ordinary skill would be able to implement the variation;

vii. If, at the time of the alleged invention, there existed a known problem for which there was an obvious solution encompassed by the patent's claims

viii. Known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations would have been predictable to one of ordinary skill in the art; and

ix. Some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

36. I have been informed and understand that secondary considerations of non-obviousness may include: (1) whether the invention proceeded in a direction contrary to accepted wisdom in the field; (2) whether there was a long felt but unresolved need in the art that was satisfied by the invention; (3) whether others had tried but failed to make the invention; (4) whether others copied the invention; (5) whether the invention achieved unexpected results; (6) whether the invention was praised by others; (7) whether others have taken licenses to use the invention; (8) whether experts or those skilled in the art at the making of the invention expressed surprise or disbelief regarding the invention; (9) whether products incorporating the invention have achieved commercial success that is attributable to the invention; and (10) whether or not others having ordinary skill in the field independently made the claimed invention at about the same time the inventor made the invention.

37. I also understand that for any such secondary consideration to be relevant, the patentee must establish a connection or nexus between the secondary consideration and the claimed invention. For example, commercial success is relevant to obviousness only if the success of the product is related to a feature of the patent claims. If, however, commercial

success is due to advertising, promotion, salesmanship or the like, or is due to features of the product other than the claimed invention, then any commercial success should not be considered an indication of non-obviousness.

38. I understand that central to the process of understanding the disclosures in a patent and assessing the validity of a patent is the notion of a person of ordinary skill in the art.

39. I understand that a person having ordinary skill in the art is a hypothetical person who analyzes the prior art without the benefit of hindsight. A person of ordinary skill in the art is presumed to be one who thinks along the lines of conventional wisdom in the art.

40. I understand that the hypothetical person of ordinary skill is presumed to have knowledge of all references that are sufficiently related to one another and to the pertinent art, and to have knowledge of all arts reasonably pertinent to the particular problem that the claimed invention addresses.

VI. SUMMARY OF OPINIONS

41. It is my opinion that the asserted claims of the '526 patent are invalid as anticipated and/or rendered obvious in light of the prior art, for at least the reasons set forth in this report.

42. I have based my opinions on the (1) documents, materials, products, and information listed in Exhibit A and in this Expert Report; (2) my professional experience; (3) my education; (4) my professional judgment; (5) my personal knowledge; (6) my appreciation of the ordinary level of skill in the art at the time of the priority date of the '526 patent; and (7) my understanding of applicable legal standards.

43. I reserve all right to amend or supplement my opinions based on further information, claim constructions, documents, discovery, reports, and/or deposition testimony

provided in this case. I reserve the right to respond to any rebuttal opinions on invalidity, or any theories of validity of which I am not currently aware.

VII. TECHNICAL BACKGROUND

44. The '526 patent, attached as Exhibit D, discloses "a processor based system having a parser is configured for validating a generic command received from a user relative to a command parse tree" and "issu[ing] a prescribed command for a selected one of the management programs according to the corresponding command format"—*i.e.*, a system that allows a user to enter a generic command which it translates into the format required by a specific management program. Ex. D, Abstract. I understand that the '526 patent was filed on June 28, 2000. The alleged invention's claimed benefit is that a user may learn a single generic command language, which the parser-translator converts to a properly formatted command for a given management tool, thus saving the user from learning the specific syntaxes and formats for multiple management programs. *Id.* at 1:31-37. As the '526 patent explains, "the new syntax provides a generic instruction set that provides an abstraction of the tool-specific command formats and syntax, enabling a user to issue command [sic] based on the relative functions, as opposed to the specific syntax for a corresponding tool 18." *Id.* at 3:31-35; *see also id.* at 7:1-9:16 (listing "Generic Command Examples").

45. Below I provide a brief overview of the relevant technology of which one ordinary skill in the art would have been aware at the time of the alleged invention.

A. **Parsers and Parse Trees**

46. A parser is used to discover the syntactic structure of a phrase or sentence or command, according to the rules of some language. For example, an English language parser might parse the sentence "Ralph bit a brown dog" into a Noun (Ralph) followed by a Verb (bit)

followed by a Noun Phrase, which itself contains an Article (a), an Adjective (brown), and a Noun (dog). Once so parsed, the English sentence might then, for example, be translated into another language.

47. A computer command line parser might parse the command "watch tcp connections" into a Command Word (watch) followed by a Protocol Type (tcp) followed by a Parameter (connections). Like all parsers, it operates by following the rules of the language (its "grammar"), which, depending on the complexity of the language, may be expressed as a table, a tree, or some other data structure. English language parsers are quite complex, while computer command line parsers are quite simple. The parsing of computer languages has been studied for decades, and numerous formalisms and methods proposed.

48. Chomsky's foundational work on formal languages in the 1950's proposed a tree representation:

> Every [finite-state, or regular] language can be represented in the form of a *tree*. At the root of the tree is a point from which all sentences start. Each word that can occur as the initial word in some sentence is represented by a branch leaving this initial point. At the end of the branch representing any particular word will be another set of branches representing each of the words that can follow the first. This process of arborization continues until every possible sentence is represented by some path through the tree; or, if the language contains an infinite set of sentences, the tree will continue indefinitely.

Noam Chomsky & George A. Miller, *Finite State Languages*, 1 Information and Control 91 (1958) ("Chomsky & Miller") at 94.

49. One of skill in the art would have known that a language that has a finite number of possible commands is a "regular language" (recognizable by a regular expression or a finite-state machine), which is at Level 3 of Chomsky's well-known hierarchy of languages. *See generally* Noam Chomsky, *On Certain Formal Properties of Grammars*, 2 Information and

Control 137 (1959) ("Chomsky"). Such languages can be represented by a finite tree structure, as proposed by Chomsky and Miller.

50. Parsers of regular command languages process commands left-to-right, at each step consulting a data structure such as a table, *see, e.g.*, Alfred V. Aho, et. al., Compilers: Principles, Techniques, and Tool (1986) ("Aho") at 126-27[1], or a tree, *see, e.g.*, Chomsky & Miller at 94, to see what command words are acceptable next choices. As explained, for example in U.S. Patent No. 6,134,709 ("Pratt"), applied for in June 1998, in its "Background of the Invention" section, "[t]he list of acceptable commands can be stored in a tree structure to reduce the space required to store all of the acceptable commands. The root nodes of the tree can contain acceptable first tokens in a command. . . .The tree can be searched to match the submitted command, token by token." Pratt at 1:40-57. Methods that use a tree representation of the language would proceed as follows. At the top of the tree (the "root"), where the parse begins, there are one or more branches that lead to nodes, each representing one valid first command word. The nodes may extend to additional branches, leading to other nodes, each representing a subsequent valid command word in light of the word or words that have already been validated. A node that has no branches is called a "leaf."

51. A parser evaluates statements according to a set of syntax rules, known in the art as a "grammar." These syntax rules define valid sequences of words (sometimes called "tokens") against which the parser compares the incoming statements. A "command parse tree" is one representation of the syntax rules in a command language grammar, but depending on the complexity of the language, such syntax rules can be expressed in other ways, such as a table, as I discussed above. These representations would be obvious to a person of skill in the art .

[1] ARISTANDCA00001453

52. For example, below I have listed one way the syntax rules represented in the command parse tree of Figure 2 of the '526 patent can be expressed in the form of rule specifications. A person of skill in the art would call these rule specifications "phrase structure rules" (a term originated by Chomsky) or "productions,"

RULE1 → watch tcp connections

RULE2 → watch udp connections

RULE3 → get tcp connection info

RULE4 → set tcp connection info

These phrase structure rules, or productions, may be associated with some action that the system should perform.

B. Command Translation and Translators

53. The notion of translating an input, such as a command expressed in some user-friendly format, into some other format, such as a command for another system component, possibly in a less user-friendly format, using a parser and translator, is fundamental to computer science and engineering. Examples abound:

- *Compilers* parse and translate high-level language statements into machine language. Aho at 1 ("[A] compiler is a program that reads a program written in one language—the *source* language—and translates it into an equivalent program in another language—the *target* language . . . .").
- *Assemblers* parse and translate human-readable (and sometimes human-generated) machine level instructions into actual binary machine code. Aho at 17-18 ("Assembly code is a mnemonic version of machine code, in which names are used instead of binary codes for operations, and names are also given to memory addresses…In the second pass [of a two-pass assembler], the assembler

scans the input again. This time, it translates each operation code into the sequence of bits representing that operation in machine language . . . .").

- *Command languages* parse and translate high-level commands typed by the user into operating system instructions. *Tops-20 User's Guide* ("Tops-20")[2] at 2.1 ("A TOPS-20 command is an instruction that specifies the function you want the TOP-20 operating system to perform. By giving TOPS-20 commands you accomplish your work through the operating system…[An] argument further identifies the information the system needs to process the command…Each part of a TOPS-20 command or subcommand is referred to as a field and is separated from each adjacent field by a space.")

C. **Command Completion**

54. In its Infringement Contentions for the '526 patent, Cisco has evidently read the patent's claims to encompass "command help" and "command-completion" functionality. *See, e.g.*, Cisco's Amended Infringement Contentions, Exhibit 526-01 (Dec. 14, 2015), at 19, 25, 33-37. Such functionality was well known years before Cisco applied for the '526 patent. For example, the TOPS-20 CLI, which was sold by the Digital Equipment Corporation in the 1980s, provided a "command help" feature that listed all possible commands beginning with a given prefix if the user typed "?" after the prefix:

> When typing a question mark, you are not limited to just one letter;
> you may type as many as you need.
>
> ```
> @CON? Command, one of the following:
>    CONNECT      CONTINUE
> ```

TOPS-20 at 2.2.

[2] ARISTANDCA00009216

55. Many prior art command line interfaces included similar help features. *See, e.g.*, U.S. Patent No. 6,724,408 ("Chen")[3] at 6:29-7:5; U.S. Patent No. 5,832,224 ("Fehskens")[4] at 23:49-24:37; JUNOS Internet Software Configuration Guide: Installation and System Management, Release 4.0 ("JUNOS Config Guide")[5] at 50.

56. Additionally, many CLIs allowed the user to type only the prefix of a desired command instead of requiring the full command. The TOPS-20 CLI supported command completion. "To give a command using abbreviated input, type only enough of the command to distinguish it from any other command. Usually, typing the first three letters is sufficient to distinguish one command from another." TOPS-20 at 2.4.3. Similarly, Avaya's Definity Audix CLI allowed users to enter partial commands and parameters: "You may abbreviate most commands. For example, you could abbreviate **change system-parameters sending-restrictions** as **ch sy s**. You may abbreviate the verb, object, or qualifier(s) by typing the first few letters of each word, in the correct spelling order." *Avaya Definity Audix System Release 4.0, Screens Reference* ("Definity Audix")[6] at 1-7. Notably, Cisco's own prior art products allowed the CLI user to enter abbreviated commands that the parser would complete. "When parsing a command, the command-line parser checks the entire parse tree, searching all alternates for a given token, in order to detect ambiguous input that might occur when abbreviated keywords are used." Cisco IOS Programmer's Guide/Architecture Reference, Software Release 12.0 ("IOS Guide")[7] at 23-1.

[3] ARISTANDCA00000874

[4] ARISTANDCA00002286

[5] ARISTANDCA00002939

[6] ARISTANDCA00000447

[7] ARISTANDCA00000893

D. Management Programs

57. The claims of the '526 patent are directed to issuing translated commands to what it calls "management programs." The '526 patent identifies several general examples of management programs, including "Operating Administration and Monitoring (OAM) tools," which are said to be "software-based resources used as administration and/or diagnostic tools for complex processor-based executable software systems, such as software-based unified messaging software systems." '526 patent, at 1:11-15. Other examples of OAM tools are provided, including "Real Time Monitoring (RTM) programs" (*id.* at 1:15-16), "Simple Network Management Protocol (SNMP) agents or scripts" (*id.* at 1:25-26), and "external binary files that execute in response to a procedure call" (*id.* at 1:24-25).

58. Such programs were well known in the art before the '526 patent. Indeed, Cisco's own prior-art products permitted users to invoke "Real Time Monitoring (RTM) programs" through a command line interface. For example, Cisco IOS 12 CLI offered a "ping" command that would permit the monitoring of a remote computer on a network. IOS Guide at 23-10. Similarly, the Avaya's Definity Audix system would perform the UNIX ping command in response to a command entered into the CLI. Definity Audix at 3-71.

VIII. OVERVIEW OF THE '526 PATENT

59. The '526 patent describes an alleged improvement for controlling complex administration and/or diagnostic software tools in processor-based systems. '526 patent at 1:11-14. It explains that, typically, each of these administration and diagnostic tools has its own command format, which system administrators must remember. *Id.* at 1:31-34. Moreover, system administrators may find it difficult "to determine which tool is the best tool (and/or which is the best syntax) to use for a given problem." *Id.* at 3:16-20.

60. To address these problems, the patent proposes a set of "generic commands" that a user can input into a parser-translator system, which will issue a corresponding "prescribed command" to a management program in that program's command format. *See id.* at Abstract. As the patent explains, "the new syntax provides a generic instruction set that provides an abstraction of the tool-specific command formats and syntax, enabling the user to issue command[s] based on the relative functions, as opposed to the specific syntax for a corresponding tool . . . ." *Id.* at 3:31-35; *see also id.* at 7:1-9:16 (listing "Generic Command Examples").)

61. The generic command is validated by the system in relation to what the patent calls a "command parse tree." *Id.* at 1:48-51. Figure 2, which is reproduced below, depicts the command parse tree (structure 22) of the preferred embodiment:

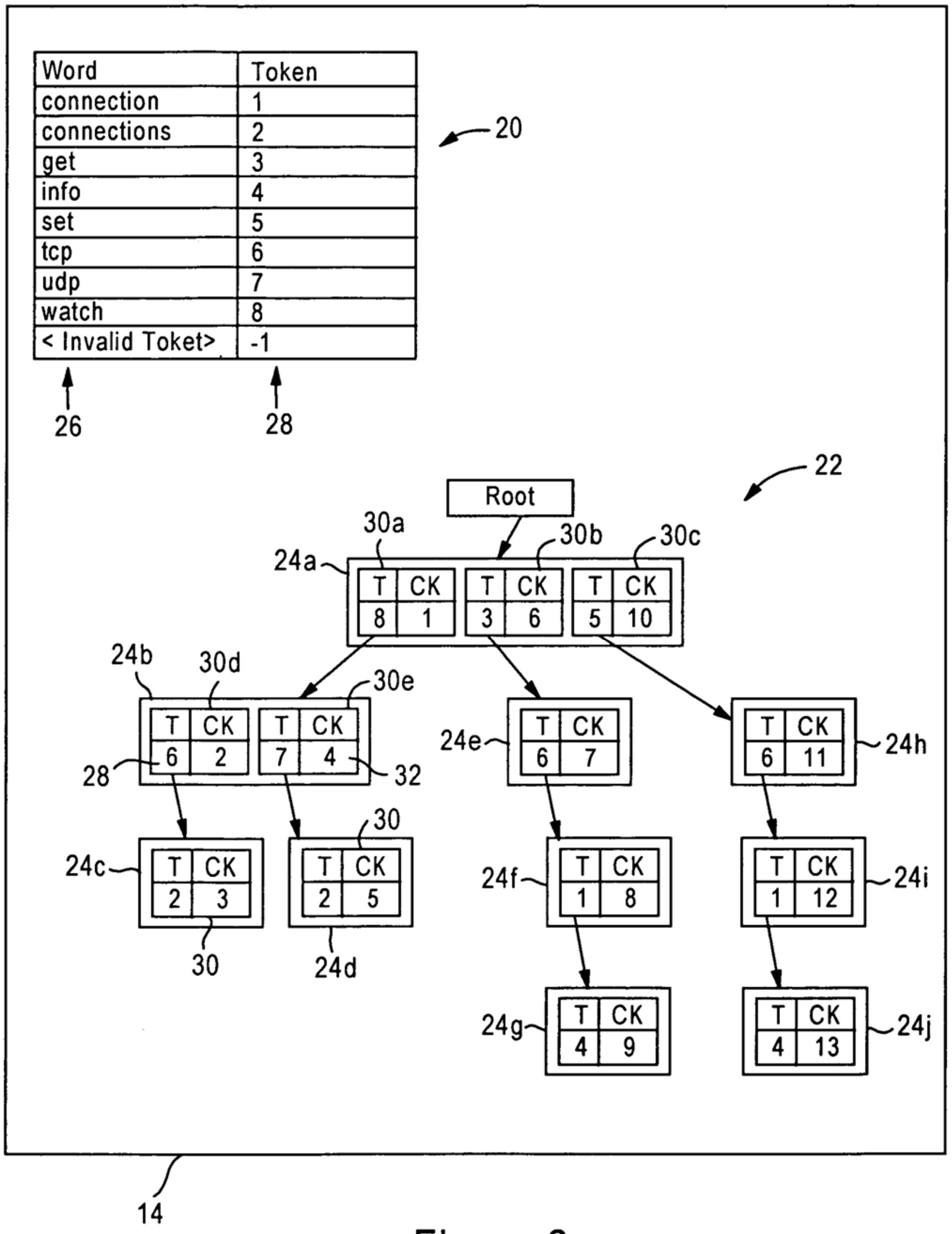


Figure 2

62. Figure 2 shows a hierarchical data representation that includes multiple elements (structures 24a-j). Each element contains at least one "token" (abbreviated "T") and a corresponding "command key" (abbreviated "CK"). *Id.* at 3:51-53. Figure 2 also depicts a

"command word translation table" (structure 20) that matches words within generic commands to tokens, which are shown as numerical values—for example, the generic-command word "watch" corresponds to the token "8." *Id.* at 3:38-46 and Fig. 2. These tokens are used to traverse the command parse tree and identify the element that's the "best match" for the received generic command. *Id.* at 3:47-51.

63. Tokens are matched with command-parse-tree elements based on the order of the words in the received generic command. *Id.* at 4:3-7, 4:14-16. Beginning at the highest level of the tree, the parser determines if the token corresponding to the first generic-command word is valid. *Id.* at 4:3-13, Fig. 3. If so, the parser traverses the next level of elements that are allowed to follow the just-validated token to determine if the next token in sequence is valid at that level. *Id.* at 4:13-16, 4:19-27. Thus, for example, referring to Figure 2, if the first two words in the received generic command are "watch tcp," the parser determines if the token corresponding to "watch" (the number "8") is valid in element 24a, then determines if the token corresponding to "tcp" (the number "6") is valid in element 24b, which depends from token "8" in element 24a.

64. This process repeats until either no tokens remain, in which case the element corresponding to the final token is identified as the "best match," or the next token in the sequence is found to be invalid, in which case the element corresponding to the last valid token is identified as the "best match." *Id.* at 4:27-36, 4:46-49. The parser then uses the "command key" that corresponds to the "best match" element to identify a "prescribed command" to issue to a selected management program. *Id.* at 3:51-54, 4:31-36, 4:49-51. For example, referring to Figure 2, if the parser receives the generic command "watch tcp connections," it will identify 24c as the "best match," and use the command key "3" to identify a "prescribed command" to issue to a selected management program.

IX. INVALIDITY ANALYSIS OF THE ASSERTED CLAIMS OF THE '526 PATENT

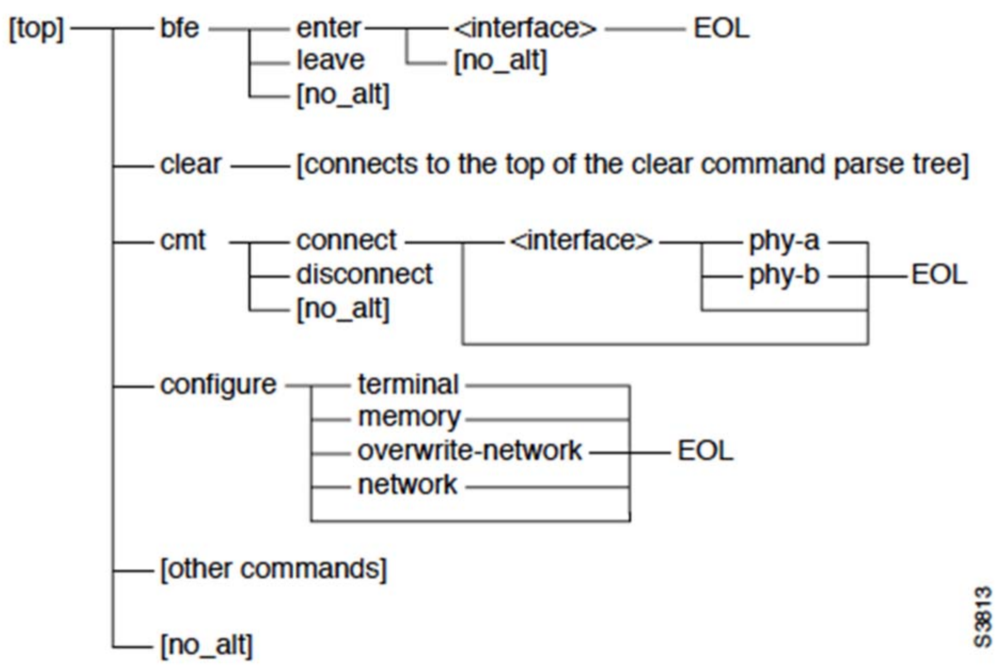
65. For the reasons explained in this report, my opinion is that each of the Asserted Claims of the '526 patent is invalid in light of the following prior art: (A) Cisco IOS 12.0; (B) Lucent Avaya Definity Audix System Release 4.0; (C) Juniper Networks' JUNOS Software Release 4.0; and (D) U.S. Patent No. 6,523,172 to Martinez-Guerra, et al.

A. **CISCO IOS 12**

66. I understand that Cisco's IOS 12 CLI ("IOS 12"), as described in Cisco IOS Programmer's Guide/Architecture Reference, Software Release 12.0, 5th Ed. (Feb. 1999) ("IOS Guide"), was in public use and on sale in the United States by February 1999, which is more than one year before the filing date of the '526 patent. *See* Bettadapur Tr. at 11:6-11, 12:18-13:12. Accordingly, I understand that the IOS 12 system is prior art under 35 U.S.C. § 102(b).

67. Cisco's "internetwork operating system" (known as "IOS") is the operating system that has been used by Cisco in its routers for over twenty years. *See, e.g.*, U.S. Patent No. 7,953,886 at 1:26-34 (Exhibit E). Network engineers can access and configure a Cisco router using commands entered via IOS's command line interface (CLI). *Id.* at 1:14-16. The CLI includes a parser, which is described as "a finite state machine described by a series of macros that define the sequence of a command's tokens," wherein "[e]ach macro defines a node in the state diagram of a command." IOS Guide at 1-9. Commands accepted by the parser are represented in a command parse tree, such as that illustrated below:

Figure 23-1 Traversing the ParseTree



Id. at 23-2.

1. Cisco IOS 12 Anticipates Claims 1 and 14

[1A] A method in a processor-based system configured for executing a plurality of management programs according to respective command formats, the method comprising:

68. In my opinion, IOS 12 practiced the claimed method in a processor-based system configured for executing a plurality of management programs according to respective command formats at least under Cisco's construction of the phrase "management programs."

69. IOS 12 is a computer operating system that runs on a "processor-based system," such as a Cisco router. *See, e.g.*, IOS Guide at 22-1 (referring to "Cisco IOS code, which assume[s] a certain type of microprocessor"); *see also id.* at 22-8, E-9 ("RP"). As the IOS Guide explains, "[t]he Cisco IOS software provides facilities for network management from the command line and management applications." *Id.* at 1-9. These network management facilities

and applications are executed in response to parsing, via a command parse tree, a command received from the IOS CLI. As described in greater detail below, upon reaching the final element of a parsed command, the software is able to identify "a pointer to a command function that is to be called to execute the command." *Id.* at 23-2. This executed command corresponds to one of a plurality of "facilities for network management . . . and management applications," which are "management programs." *Id.* at 1-9.

70. Accordingly, it is my opinion that IOS 12 practiced "[a] method in a processor-based system configured for executing a plurality of management programs according to respective command formats."

[1B] receiving a generic command from the user;

71. In my opinion, IOS 12 practiced this limitation of claim 1 under at least Cisco's construction of the phrase "generic command."

72. As explained in the IOS Guide, IOS 12's "[p]rocesses respond to commands entered at terminals and perform tasks for users." *Id.* at 1-2. The IOS Guide provides several examples of generic commands that can be entered by a user and that are received by the CLI parser of IOS 12. One such example is "**configure network**," which the IOS Guide discusses in the context of the command parse tree of Figure 23-1. *Id.* at 23-2. "**[C]onfigure network**" is described as the "input," which in this context refers to the user input to the command line interface. *Id.*; *see also id.* at xlvi ("Boldface indicates commands and keywords that are entered literally as shown."). In another example, the IOS Guide discusses the process of parsing "the user input [that] begins with the word **disable** . . . ." *Id.* at 23-3. These are both "generic commands" at least under Cisco's proposed construction.

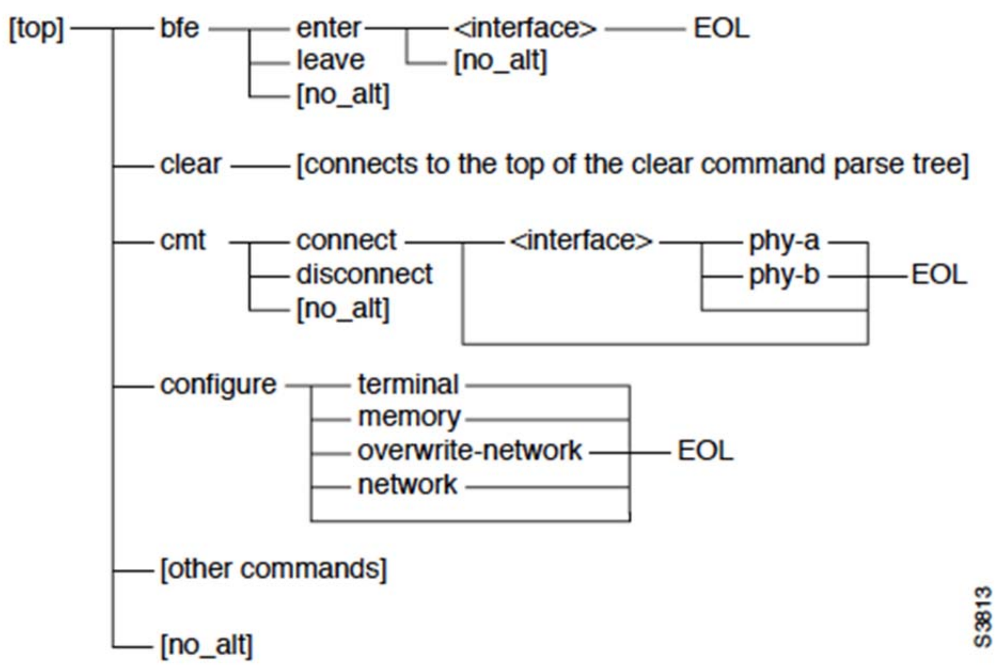
73. Accordingly, it is my opinion that IOS 12 practiced "receiving a generic command from the user" under at least Cisco's proposed construction of "generic command."

> *[1C] "validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format,"*

74. I understand that Cisco's proposed construction of "command parse tree" requires, in part, "a hierarchical data representation." *See* Ex. C, at 4. In my opinion, IOS 12 included a hierarchical data representation, as I explain below. In addition, I understand that Cisco's proposed construction of "command parse tree" requires "having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value." *See id.* To the extent Cisco reads this limitation as *not* requiring every single element of the tree to have a corresponding at least one command action value, it is my opinion that IOS 12 practiced this claim limitation for the reasons that follow. Additionally, it is my opinion that, at least under Arista's proposed construction of "command parse tree," IOS 12 practiced this claim limitation.

75. IOS 12 performed the step of validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format. Figure 23-1 of the IOS Guide, for example, "shows a global view of the parse tree for a subset of the Cisco IOS router EXEC commands." *Id.* at 23-1.

Figure 23-1 Traversing the ParseTree



76. As the Guide explains, "the accepting nodes are distributed to the right, and alternate nodes are distributed down the figure on the left." *Id.* Using this command parse tree, IOS 12 validates the received generic command. As the IOS Guide further explains, "[w]hen parsing a command, the command-line parser checks the entire parse tree, searching all alternates for a given token, in order to detect ambiguous input that might occur when abbreviated keywords are used." *Id.* Specifically, "[w]hen traversing the parse tree, the parser does the following: . . . The parser begins at [top] and compares the first input token with all the alternates listed on the left, starting with *bfe*, *clear*, and so on. . . . When a match occurs, the parser transitions to the accepting node on the right. . . . The parser searches the remaining alternate nodes to identify ambiguous commands. . . . After the entire parse tree has been searched, if there has only been one command match, the saved CSB is restored and the specified command function is called with the CSB as its argument." *Id.* One of ordinary skill in the art

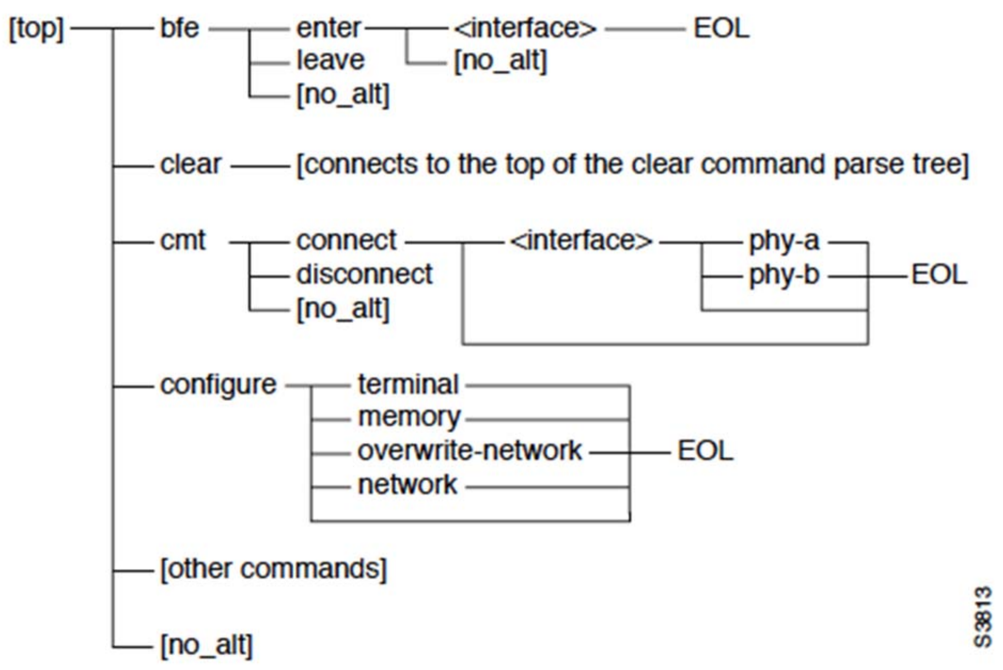
reviewing this disclosure would understand that the parse tree of Figure 23-1 specifies valid generic commands relative to a prescribed generic command format because the IOS Guide describes the parser comparing input tokens to the nodes of the command parse tree and identifying matches. In doing so, the parser validates the generic command received from the user. *See also id.* at 23-2 (discussing the specific example of the generic command "configure network").

> *[1D] the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value,*

77. As I explained with regard to the previous claim element, I understand that Cisco's proposed construction of "command parse tree" requires "a hierarchical data representation having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value." To the extent Cisco reads this limitation as *not* requiring every single element of the tree to have a corresponding at least one command action value, it is my opinion that IOS 12 practiced this claim limitation for the reasons that follow.

78. As discussed with regard to the preceding limitation, IOS 12 included a command parse tree, a portion of which is illustrated in Figure 23-1. The IOS Guide's description of the command parse tree further discloses that it has elements each specifying at least one corresponding generic command component.

Figure 23-1 Traversing the ParseTree



79. For example, the IOS Guide explains that "configure network" is a potential input received by the command line interface. *Id.* at 23-2. In the command parse tree of Figure 23-1, both "configure" and "network" are what the '526 patent refers to as "generic command components." The first input token of the command, "configure," is displayed as a node just below the [top] level of the command parse tree. *See* Fig. 23-1. The second input token, "network," is one of four nodes that may be accepted from the token "configure." *See id.*; *see also id.* ("When the parser reaches **configure**, which matches, the parser advances the input pointer and checks the tokens accepted from **configure**, which are listed to the right of **configure**.").

80. The IOS Guide also discloses that each node or element of the command parse tree includes a corresponding at least one command action value. In the "configure network" example, the IOS Guide explains that once the keyword "network" is matched, that is "followed

by a match of end of line (EOL)." *Id.* at 23-2. "When EOL is reached, the state of the parser is stored on a stack. This state includes parsed values ***and a pointer to a command function that is to be called to execute the command***." *Id.* (emphasis added). In other words, once the EOL is reached—which indicates that preceding input word (e.g., "network") was the final word input by the user—the parser uses the pointer corresponding to the final matched token to identify a command function that is to be called to execute the command. This is what the '526 refers to as "a corresponding command action value."

[1E] the validating step including identifying one of the elements as a best match relative to the generic command; and

81. In my opinion, IOS 12 practiced this limitation of claim 1.

82. As discussed in the preceding limitation, IOS 12 validated a generic command, such as "configure network," by parsing each token of the input and identifying one of the elements of the command parse tree as a best match relative to the generic command. The IOS Guide explains that once the keyword "network" is matched, followed by the match of the end of line (EOL), "the state of the parse is stored on a stack. This state includes parsed values and a pointer to a command function that is to be called to execute the command." *Id.* at 23-2. One of ordinary skill in the art viewing these disclosures of the IOS Guide would understand that the final node or element of the command parse tree, which indicates EOL, is what the '526 patent refers to as a "best match" for the generic command "configure network." This is because matching the EOL indicates that a complete, valid generic command was received and parsed.

83. Accordingly, it is my opinion that IOS 12 practiced "the validating step including identifying one of the elements as a best match relative to the generic command."

[1F] issuing a prescribed command of a selected one of the management

> *programs according to the corresponding command format, based on the identified one element.*

84. In my opinion, IOS 12 practiced this limitation of claim 1.

85. As discussed above, the IOS Guide discloses that IOS 12 identified an element of the command parse tree that was a "best match" for the generic command received from the user. In the case of the "configure network" command, once the EOL of the generic command is matched with the corresponding element in the command parse tree of Figure 23-1, "the state of the parse is stored on a stack" and "the state includes parsed values and a pointer to a command function that is be called to execute the command." *Id.* at 23-2 (emphasis added); see also id. at 23-1 ("After the entire parse tree has been searched, if there has only been one command match, the saved CSB is restored and the specified command function is called with the CSB as its argument. The command function extracts the parsed values from the CSB.").

86. One of ordinary skill in the art viewing the IOS Guide's disclosures would understand that the reference to a "command function that is to be called to execute the command" refers to the prescribed command of a selected management program, as the Guide describes the operation of the CLI parser and command parse tree in the context of "Management Services." *Id.* at 1-9; *see also generally id.* at Chapter 23 (entitled "Management Services").

87. Moreover, to the extent that the Court is persuaded that the term "management programs" requires "separate tools or external agents," it would have been obvious to use IOS 12 to issue a prescribed command to separate tools or external agents in light of the understanding of one of skill in the art. For example, as discussed in Part IX.B, below, the Definity Audix system would perform a UNIX command in response to a separate command entered into the

system's CLI. It would have been obvious likewise to adapt IOS 12 to issue commands to a "separate" environment, which might use different command formats.

88. Accordingly, it is my opinion that IOS 12 practiced "issuing a prescribed command of a selected one of the management programs according to the corresponding command format, based on the identified one element" under either proposed construction of "management programs."

Claim 14

[14A] A computer readable medium having stored thereon sequences of instructions for executing a plurality of management programs according to respective command formats, the sequences of instructions including instructions for performing the steps of:

[14B] receiving a generic command from the user;

[14C] validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format, the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating step including identifying one of the elements as a best match relative to the generic command; and

[14D] issuing a prescribed command of a selected one of the management programs according to the corresponding command format, based on the identified one element.

89. Claim 14 is substantially identical to claim 1, which I address above, except that it is drafted as a computer readable medium storing instructions for performing the method claimed in claim 1. One of ordinary skill in the art would understand from the IOS Guide, which describes aspects of the operation of IOS 12, that IOS 12 consisted of instructions for performing the claimed method and that such instructions were stored on a computer readable medium, such

as RAM, ROM, and/or a disc. Accordingly, for the same reasons set forth as to claim 1, it is my opinion that IOS 12 anticipates claim 14.

2. Cisco IOS 12 Renders Obvious Claims 6, 10, 11, 13, 15, 16, 19 & 23

a. Non-Asserted Claim 2

90. I understand that claim 2 is not currently asserted against Arista, but claim 6, which depends from claims 2-5, is asserted. Because I understand that the limitations of claim 2 are thus part of claim 6, I will address it below. In my opinion, claim 2 is rendered obvious by IOS 12.

> *[2A] The method of claim 1, wherein the generic command includes at least one input command word, the validating step including:*

91. As explained above, IOS 12 discloses, for example, the generic command "configure network," which includes at least one input command word, "configure." *Id.* at 23-2. Accordingly, it is my opinion that IOS 12 discloses that "the generic command includes at least one input command word."

> *[2B] comparing each input command word to a command word translation table, configured for storing for each prescribed command word a corresponding token, for identification of a matching token; and determining a presence of the matching token within the command parse tree for each input command word.*

92. As discussed above with regard to claim 1, the Guide explains how the IOS 12 CLI, when parsing a command, looks within the command parse tree for the presence of tokens that match each input command word. *See id.* at 23-1. In my opinion, it would have been obvious, based on the understanding and knowledge of one of skill in the art, to use a command word translation table for the purpose of matching input command words to tokens. A person of skill in the art would have found the use of such a table to be an obvious implementation choice

that could reduce the size of the command parse tree (numerical tokens can be smaller that command words). Indeed, for the IOS 12 parser to work as described, the system would need some mechanism for correlating command words to corresponding tokens. In my opinion, the use of a "table" for that purpose was obvious.

b. Non-Asserted Claim 3

93. I understand that claim 3 is not currently asserted against Arista, but claim 6, which depends from claims 2-5, is asserted. Because I understand that the limitations of claim 3 are thus part of claim 6, I will address it below.

> *[3] The method of claim 2, wherein the determining step includes recursively traversing the command parse tree based on an order of the input command words for identification of the matching token within the identified one element.*

94. I come now to the phrase "recursively traversing." I agree with Cisco's expert Dr. Kevin C. Almeroth, who opined on this in his declaration in support of Cisco's claim construction brief: " 'Recursively traversing [a]…tree' is a concept that is well known in the art[,]" and further opined that "[i]n the art, traversing a tree is a high-level concept that describes ***systematically processing each element of the tree***." Almeroth Decl., ¶¶ 71, 75 (emphasis added),. His explanation of "recursively" was less clear, in part because he relied on an obscure paper that a person of skill at the time would be unlikely to have read, since it was published roughly eight years after the '526 patent's priority date. Almeroth Decl., ¶ 80.

95. A person of skill in the art would immediately understand that "recursively traversing" a tree meant using a computer procedure or function that calls itself (on subtrees) with the purpose of visiting every single node in a tree. The '526 patent's teachings are of course nowhere near this. So we must allow the '526 patentees to establish an alternate meaning

for this phrase. In my opinion, one of ordinary skill in the art would understand the term "recursively traversing" in the '526 patent to mean searching (in the tree) using a process that repeats itself. Under this understanding, IOS 12 recursively traverses a parse tree. As the IOS Guide explains, "[w]hen parsing a command, the command-line parser checks the entire parse tree, searching all alternates for a given token, in order to detect ambiguous input that might occur when abbreviated keywords are used." IOS Guide at 23-1.

96. The Guide further explains that, when traversing the parse tree, the parser does the following:

> **1** The parser begins at [top] and compares the first input token with all the alternates listed on the left, starting with *bfe*, *clear*, and so on.
>
> **2** When a match occurs, the parser transitions to the accepting node on the right.
>
> **3** The parser allocates a new console status block (CSB).
>
> **4** The parser searches the remaining alternate nodes to identify ambiguous commands.
>
> **5** After the entire parse tree has been searched, if there has only been one command match, the saved CSB is restored and the specified command function is called with the CSB as its argument. The command function extracts the parsed values from the CSB.

IOS Guide at 23-1.

97. The IOS Guide's discussion of traversing the command parse tree is consistent with the '526 patent's description of "recursively traversing" the command parse tree because, for each input word, it searches successive levels of the tree to determine the best match. *Id.* ("When parsing a command, the command-line parser checks the entire tree, searching all alternates for a given token . . . . When generating nonvolatile output, which builds an expression that describes how a platform is configured, the entire parse tree must be traversed to accurately

reflect the current state of the platform."); see also *id.* (enumerating five steps performed by the parser "[w]hen traversing the parse tree").

98. Accordingly, it is my opinion that IOS 12 practiced "the method of claim 2, wherein the determining step includes recursively traversing the command parse tree based on an order of the input command words for identification of the matching token within the identified one element."

99. Accordingly, it is my opinion that, under Dr. Almeroth's professed understanding of "recursively traversing," IOS 12 practiced "recursively traversing the command parse tree based on an order of the input command words for identification of the matching token within the identified one element."

c. Non-Asserted Claim 4

100. I understand that claim 4 is not currently asserted against Arista, but claim 6, which depends from claims 2-5, is asserted. Because I understand that the limitations of claim 4 are thus part of claim 6, I will address it below.

> *[4] The method of claim 3, wherein the issuing step includes issuing the prescribed command based on a corresponding command key specified for the matching token within the identified one element.*

101. Dependent claim 4 introduces the limitation of a "command key" specified for the matching token. In my opinion, it would have been obvious, based on the understanding and knowledge of one of skill in the art, to issue the prescribed command based on a corresponding command key specified for the matching token.

102. As discussed above with regard to claim [1F], the IOS Guide discloses that IOS 12 issued a command after matching the final token of an input command with a corresponding node of the command parse tree. *See id.* at 23-1, 23-2. For example, in explaining the process of

parsing the generic command "configure network," the IOS Guide explains that after matching "configure" and "network" to elements of the command parse tree of Figure 23-1, a match of the "end of line (EOL)" occurs, wherein "the state of the parse is stored on a stack. This state includes parsed values and *a pointer to a command function that is to be called to execute the command*." *Id.* at 23-1.1 (emphasis added). One of ordinary skill in the art would understand from these disclosures that the "pointer" is a command key specified for the matching EOL element in the parse tree of Figure 23-1 because it is the data that corresponds to the final matching token and that tells IOS 12 which prescribed command to issue based on the parsed input command. This is consistent with the IOS Guide's explanation of how the CLI parser operates with reference to nodes of the command parse tree: "[e]ach macro defines a node in the state diagram of a command. This definition includes a pointer to the node to process if the current node matches the command-line input . . . ." *Id.* at 1-9.

103. Accordingly, it is my opinion that IOS 12 practiced "[t]he method of claim 3, wherein the issuing step includes issuing the prescribed command based on a corresponding command key specified for the matching token within the identified one element."

d. Non-Asserted Claim 5

104. I understand that claim 5 is not currently asserted against Arista, but claim 6, which depends from claims 2-5, is asserted. Because I understand that the limitations of claim 5 are thus part of claim 6, I will address it below.

> *[5] The method of claim 4, wherein the issuing step further includes accessing a prescribed translator configured for converting the generic command according to the corresponding command format into the prescribed command based on the corresponding command key.*

105. The IOS 12 CLI includes a parser, which is described as "a finite state machine described by a series of macros that define the sequence of a command's tokens," wherein "[e]ach macro defines a node in the state diagram of a command." IOS Guide at 1-9. Moreover, as explained with regard to claim 1 above, the IOS CLI parser would accept a generic command, such as "configure network". In the case of the "configure network" command, once the EOL of the generic command is matched with the corresponding element in the command parse tree of Figure 23-1, "the state of the parse is stored on a stack" and "the state includes parsed values and a pointer to a command function that is be called ***to execute the command***." *Id.* at 23-2 (emphasis added); *see also id.* at 23-1 ("After the entire parse tree has been searched, if there has only been one command match, the saved CSB is restored and the specified command function is called with the CSB as its argument. The command function extracts the parsed values from the CSB.").

106. Accordingly, it is my opinion, based on my discussion of claim terms 2 through 5, that IOS 12 rendered obvious "the issuing step further includes accessing a prescribed translator configured for converting the generic command according to the corresponding command format into the prescribed command based on the corresponding command key."

e. Claim 6

[6] The method of claim 5, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command.

107. Claim 6 is dependent upon non-asserted claim 5, which itself depends from non-asserted claims 4, 3, and 2 respectively. As discussed above, it is my opinion that IOS 12

renders obvious limitations of non-asserted claims 2-5. In my opinion, IOS 12 further renders obvious the limitations of claim 6.

108. As discussed above with regard to claims 1C and 1E, IOS 12 validates the generic command received from the user by identifying the element of the command parse tree having the best match relative to the generic command. *See* discussion of claims 1C and 1E, above; *see also* IOS Guide at 23-1 (discussing the parser process for traversing the parse tree; "if there has only been one command match, the saved CSB is restored and the specified command function is called with the CSB as its argument. The command function extracts the parsed values from the CSB."); *id.* at 23-2 (discussing the "configure network" example and disclosing that the parser matches the last valid input command word to a node of the command parse tree).

109. These examples disclose that the validating step includes validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command. In the example given, the "at least a portion of the generic command" is the entire command "configure network," which terminates with the end of line (EOL). *See id.* Because the claim specifies only a minimum threshold ("at least a portion of"), validating the entire command "configure network" discloses this limitation.

110. Even if this was not the case, the IOS Guide provides evidence that the parser validated a portion of the generic command less than the entire command. For example, in describing the parser macros, the IOS Guide states that the macro's "definition includes a pointer to the node to process if the current node matches the command-line input *and an alternative node to process regardless of whether the current node is accepted*." *Id.* at 1-9 (emphasis added). One of ordinary skill in the art would understand, in the context of the IOS Guide's discussion, that this means the parser may process the last validated token if the current token is

not accepted (e.g., if the current token is invalid). Moreover, the IOS 12 CLI parser includes the ability to parse partial commands, such as when abbreviated input command words are used. *See id.* at 23-1 ("When parsing a command, the command-line parser checks the entire parse tree, searching all alternates for a given token, in order to detect ambiguous input that might occur when abbreviated keywords are used."). To the extent "at least a portion of the generic command" extends to abbreviated commands, then IOS 12 practiced this limitation.

111. As discussed above with regard to claim 1F, IOS 12 issues a prescribed command based on the identified one element corresponding to the portion of the generic command. In the "configure network" example, once the EOL of the generic command is matched with the corresponding element in the command parse tree of Figure 23-1, "the state of the parse is stored on a stack" and "the state includes parsed values and a pointer to a command function that is be called ***to execute the command***." *Id.* at 23-2 (emphasis added); *see also id.* at 23-1 ("After the entire parse tree has been searched, if there has only been one command match, the saved CSB is restored and the specified command function is called with the CSB as its argument. The command function extracts the parsed values from the CSB."). In this case, when the EOL is matched with the corresponding element of the command parse tree of Figure 23-1, the prescribed command is issued based on the pointer corresponding to that element ("a pointer to a command function that is to be called to execute the command").

112. Accordingly, it is my opinion that IOS 12 renders obvious "[t]he method of claim 5, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command."

f. Claim 10

[10A] A system configured for executing a plurality of management programs according to respective command formats, the system comprising:

113. In my opinion, IOS 12 discloses the system of claim 10.

114. As discussed above as to the preamble of claim 1, IOS 12 is an operating system that runs on Cisco routers and that "provides facilities for network management from the command line and management applications." IOS Guide at 1-9. In parsing user-entered commands, the IOS CLI is able to identify "a pointer to a command function that is to be called to execute the command." *Id.* at 23-2. This executed command corresponds to one of the plurality of "facilities for network management . . . and management applications," which are "management programs."

115. Accordingly, it is my opinion that IOS 12 discloses "[a] system configured for executing a plurality of management programs according to respective command format."

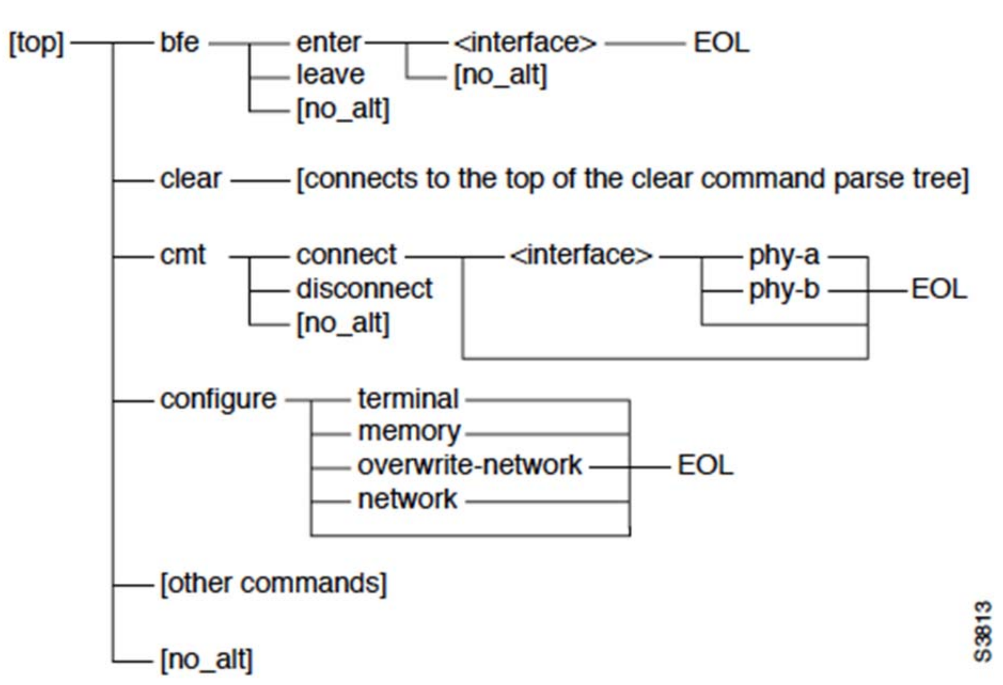
[10B] a parser having a command parse tree configured for validating a generic command received from a user, the command parse tree configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the parser identifying one of the elements as a best match relative to the generic command; and

116. In my opinion, IOS 12 discloses this limitation of claim 10.

117. As discussed above as to claim 1, IOS 12 included a "Cisco IOS command-line parser," which is "a finite state machine described by a series of macros that define the sequence of a command's tokens." *Id.* at 23-1. The described parser included a "command parse tree" for validating a generic command received from the user (e.g., "configure network," as discussed with reference to claim 1B, above): "When parsing a command, the command-line parser checks

the entire parse tree, searching all alternates for a given token, in order to detect ambiguous input that might occur when abbreviated keywords are used." *Id.* Figure 23-1, displayed below, "shows a global view of the parse tree for a subset of the Cisco IOS router EXEC commands." *Id.*

Figure 23-1 Traversing the Parse Tree



118. The remaining limitations of claim 10B repeat those recited in claim 1C-1E, which I have addressed above. Because the limitations are substantially identical, my analysis as to claims 1C-1E applies to claim 10B.

119. Accordingly, it is my opinion that IOS 12 discloses "a parser having a command parse tree configured for validating a generic command received from a user, the command parse tree configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command

component and a corresponding at least one command action value, the parser identifying one of the elements as a best match relative to the generic command."

> *[10C] a plurality of translators configured for issuing commands for the management programs according to respective command formats, the parser outputting a prescribed command to a selected one of the translators based on the identified one element.*

120. In my opinion, IOS 12 discloses this limitation of claim 10.

121. As discussed above with regard to claim 1F, IOS 12 issues a prescribed command to a management program based on the identified one element that is a best match relative to the generic command. *See* discussion above as to claim 1. For example, when the generic command "configure network" is parsed, the tokens "configure" and "network," followed by the end-of-line (EOL), are matched with nodes of command parse tree of Figure 23-1. *See id.* at 23-2. As the IOS Guide explains, "[w]hen EOL is reached, the state of the parse is stored on a stack. This state includes parsed values and a pointer to a command function that is to be called to execute the command." *Id.*

122. One of ordinary skill in the art would understand in the context of IOS 12 that "a command function that is to be called" is "a prescribed command to a selected one of the translators based on the identified element." The "command function" is output from the CLI parser based on the identified element because it is identified by the pointer corresponding to that element. *See id.* at 23-1.

123. One of ordinary skill in the art would recognize from this discussion, as well as the exemplary generic commands identified in the command parse tree of Figure 23-1, that it would be obvious to include a plurality of translators to receive the various "command functions" corresponding to the identified elements of the parse tree.

124. Accordingly, it is my opinion that IOS 12 renders obvious the limitation "a plurality of translators configured for issuing commands for the management programs according to respective command formats, the parser outputting a prescribed command to a selected one of the translators based on the identified one element."

g. Claim 11

[11] The system of claim 10, wherein the parser further comprises a command word translation table configured for storing for each prescribed command word a corresponding token for identification of a matching token, the parser configured for determining a presence of the matching token within the command parse tree for each input command word."

125. Claim 11 is substantially similar to claim 2, which I discuss above, but is drafted in terms of a system that uses a parser. As discussed above as to claim 10, IOS 12 included a parser, as claimed. The remainder of my analysis of claim 2 applies to claim 11, which I incorporate herein by reference. Accordingly, it is my opinion that IOS 12 renders obvious claim 11.

h. Non-Asserted Claim 12

126. I understand that claim 12 is not currently asserted against Arista, but claim 13, which depends from claim 12, is asserted. Because I understand that the limitations of claim 12 are thus part of claim 13, I will address it below.

[12] The system of claim 11, wherein the parser recursively traverses the command parse tree based on an order of the input command words for identification of the matching token within the identified one element.

127. Non-asserted claim 12 is substantially identical to non-asserted claim 3, which I address above. Accordingly, for the same reasons set forth as to claim 3, it is my opinion that IOS 12 rendered obvious claim 12.

i. Claim 13

[13] The system of claim 12, wherein the parser validates at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command.

128. Claim 13 is dependent upon non-asserted claim 12, which itself depends from claims 11 and 10, respectively. As discussed above, it is my opinion that IOS 12 rendered obvious all limitations of claims 10-12. In my opinion, IOS 12 further rendered obvious the limitations of claim 13.

129. Claim 13 is substantially identical to claim 6, which I address above. Accordingly, for the same reasons set forth as to claim 6, it is my opinion that IOS 12 discloses claim 13.

j. Claim 15

[15A] The medium of claim 14, wherein the generic command includes at least one input command word, the validating step including:

[15B] comparing each input command word to a command word translation table, configured for storing for each prescribed command word a corresponding token, for identification of a matching token; and

[15C] determining a presence of the matching token within the command parse tree for each input command word.

130. Claim 15 is substantially identical to claim 2, which I address above. Accordingly, for the same reasons set forth as to claim 2, it is my opinion that IOS 12 rendered obvious claim 15.

k. Claim 16

[16] The medium of claim 15, wherein the determining step includes recursively traversing the command parse tree based on an order of the input command words for identification of the matching token within the identified one element.

131. Claim 16 is substantially identical to claim 3, which I address above. Accordingly, for the same reasons set forth as to claim 3, it is my opinion that IOS 12 rendered obvious claim 16.

l. Non-Asserted Claim 17

132. I understand that claim 17 is not currently asserted against Arista, but claim 19, which depends from claims 17-18, is asserted. Because I understand that the limitations of claim 17 are thus part of claim 19, I will address it below.

> *[17] The medium of claim 16, wherein the issuing step includes issuing the prescribed command based on a corresponding command key specified for the matching token within the identified one element.*

133. Claim 17 is substantially identical to claim 4, which I address above. Accordingly, for the same reasons set forth as to claim 4, it is my opinion that IOS 12 rendered obvious non-asserted claim 17.

m. Non-Asserted Claim 18

134. I understand that claim 18 is not currently asserted against Arista, but claim 19, which depends from claims 17-18, is asserted. Because I understand that the limitations of claim 18 are thus part of claim 19, I will address it below.

> *[18] The medium of claim 17, wherein the issuing step further includes accessing a prescribed translator configured for converting the generic command according to the corresponding command format into the prescribed command based on the corresponding command*

135. Claim 18 is substantially identical to claim 5, which I address above. Accordingly, for the same reasons set forth as to claim 5, it is my opinion that IOS 12 rendered obvious non-asserted claim 18.

n. Claim 19

[19]The medium of claim 18, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command.

136. Claim 19 is dependent upon non-asserted claim 18, which itself depends from claims 17, 16, 15, and 14, respectively. As discussed above, it is my opinion that IOS 12 rendered obvious all limitations of claims 14-18. In my opinion, IOS 12 further rendered obvious the limitations of claim 19.

137. Claim 19 is substantially identical to claim 6, which I address above. Accordingly, for the same reasons set forth as to claim 6, it is my opinion that IOS 12 rendered obvious claim 19.

o. Claim 23

138. Claim 23 is substantially identical to claim 10, which I address above, except that limitation 23B is recited in "means-plus-function" form. Accordingly, in the analysis below, I discuss only claim 23B. For all other limitations, I incorporate by reference my opinion and analysis from the equivalent limitations of claim 10.

[23A] A system configured for executing a plurality of management programs according to respective command formats, the system comprising:

[23B] means for validating a generic command received from a user, the validating means configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating means identifying one of the elements as a best match relative to the generic command; and

> *[23C] a plurality of translators configured for issuing commands for the management programs according to respective command formats, the validating means outputting a prescribed command to a selected one of the translators based on the identified one element.*

139. I understand that Cisco proposes the following structure in its proposed construction of the means-plus-function term of element 23[B]: "Parser 14 in Figure 2, which includes the command word translation table 20 and the command parse tree 22." As I explained above in relation to claim 2, it would have been obvious, based on the understanding and knowledge of one of skill in the art, to use a command word translation table for the purpose of matching input command words to tokens. As for the "command parse tree," to the extent Cisco reads its claimed "validating means" as *not* requiring every single element of the tree to have a corresponding at least one command action value, it is my opinion that IOS 12 practiced this claim limitation for the reasons I explained in my discussion of claim elements 1[C] – 1[D], which I incorporate herein by reference.

B. Lucent Avaya Definity Audix System Release 4.0

140. I understand that Lucent Avaya's Definity Audix System Release 4.0 was described in the following Avaya publications dated May 1999: Avaya Definity Audix System Release 4.0, System Description Pocket Reference (May 1999) ("Audix Reference");[8] Installation and Switch Administration for the Avaya Definity Audix System Release 4.0 (May 1999) ("Audix Admin").[9]

141. Definity Audix was an enterprise-type voice messaging system that integrated with a customer's Lucent Technologies switch to provide various messaging capabilities. *See*

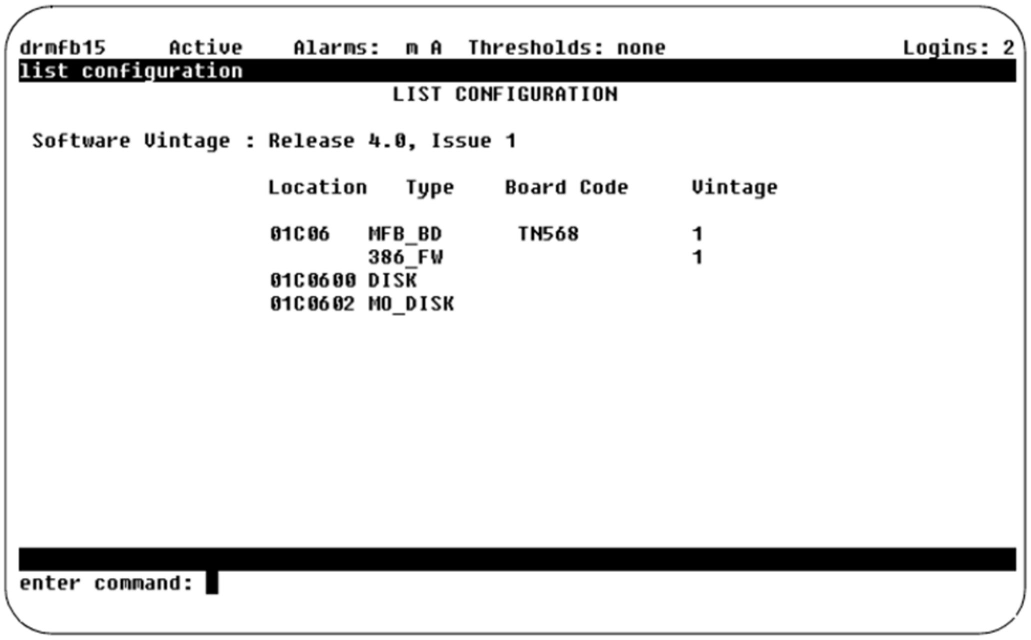
[8] ARISTANDCA00000787

[9] ARISTANDCA00000075

Audix Reference" at 1-2, 1-5. For example, the system included the ability to support up to 2000 local users, store up to 100 hours of voice messages, exchange messages with other voice messaging systems, among other abilities. *Id.* at 2-1.

142. The Definity Audix software "allows the DEFINITY AUDIX system to communicate with the switch through a telephone-like interface. This type of operation is called set-type emulation. The commands that subscribers and the switch use to access the system's software correspond to the interface on a digital telephone, or set." *Id.* at 2-3. The Definity Audix system "operates inside the customer's switch." *Id.* at 2-9; *see also id.* Fig. 2-6 (illustrating a Definity Audix system installed in a CMC switch). The Definity Audix system was configured remotely via a terminal interface, through which an administrator could log in to the Definity Audix system. *See, e.g., id.* at 2-29-31; *see also* Audix Admin at 3-1; Lucent Technologies, Getting Started Presentation, Dec. 1999 ("Getting Started")[10] at 3-5. The terminal window, including a command line interface at the bottom, is shown in the figure below:

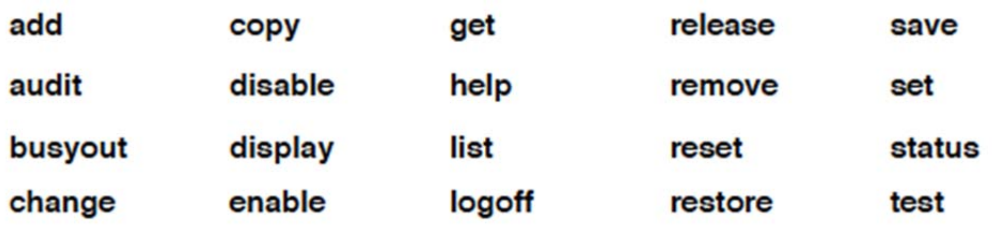
[10] ARISTANDCA00008843



```
drmfb15      Active     Alarms:  m A  Thresholds: none                     Logins: 2
list configuration
                                      LIST CONFIGURATION

 Software Vintage : Release 4.0, Issue 1

                        Location    Type     Board Code      Vintage

                        01C06    MFB_BD       TN568          1
                                 386_FW                      1
                        01C0600 DISK
                        01C0602 MO_DISK




enter command:
```

Audix Admin at 2-31.

143. Once logged into the Definity Audix system, the administrator could configure various options and parameters using the command line interface of the terminal. The administrator entered commands using a "verb-object syntax" to perform these tasks. Avaya Definity Audix System Release 4.0, Screens Reference (May 1999) ("Audix Screens") at 1-5. Definity Audix defined a set of twenty command verbs, which "specif[y] the type of action" to take in response to the command:



add	copy	get	release	save
audit	disable	help	remove	set
busyout	display	list	reset	status
change	enable	logoff	restore	test

Id. at 1-6. The command verb was followed by a command object phrase, which was usually the name of the screen that appeared on the computer display (e.g., "alarms, measurements, remote-messages, and attendants"). *Id.* The final element of the command syntax was a command-line qualifier, which was an additional parameter such as a name or extension number. *Id.* Definity Audix's command structure allowed most screens to be "activated by more than one version of a command. The different versions of the screen activation command are distinguished by the verb that begins the command line and by the qualifiers that end the command line." *Id.* For example, the following commands all activated the Subscriber screen:

add subscriber *subscriber-id*

change subscriber *subscriber-id*

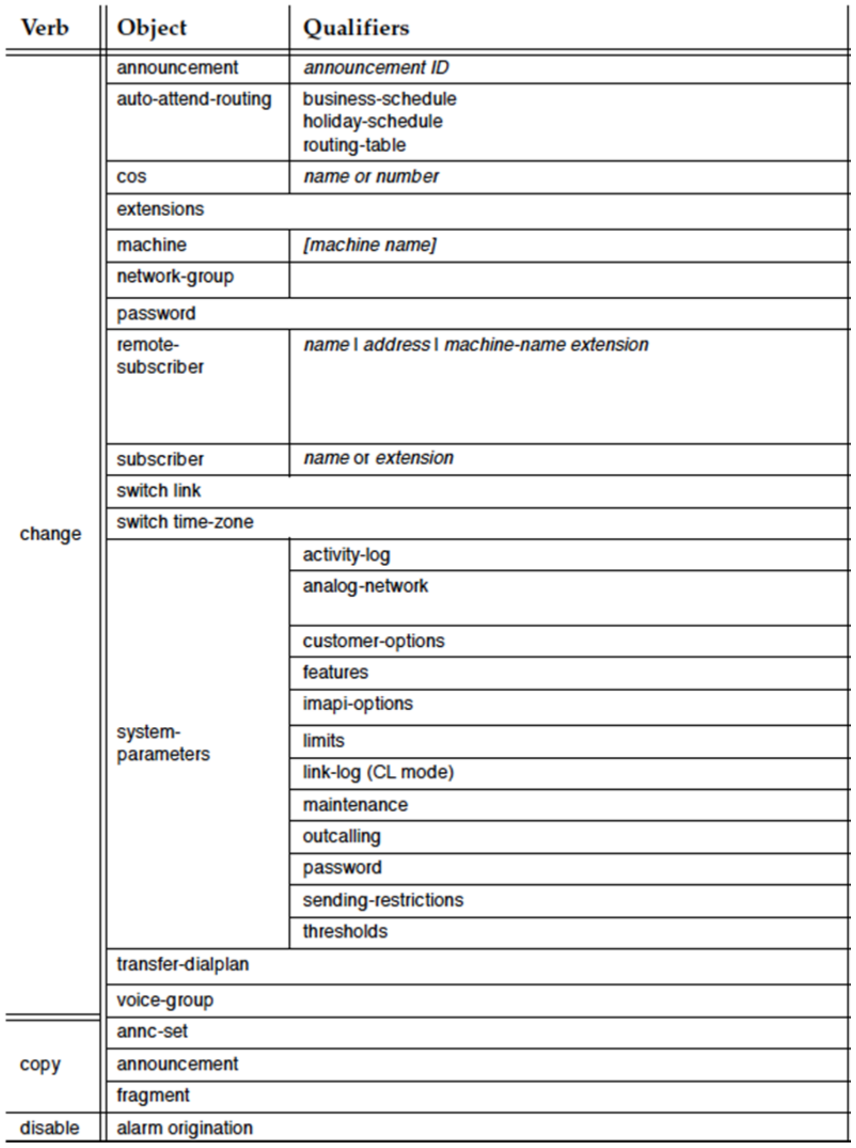
display subscriber *subscriber-id*

remove subscriber *subscriber-id*

Id. at 1-6–1-7.

144. Definity Audix included features to facilitate management of the system using these commands, including command line abbreviation and command line help. Command line abbreviation and help allowed system administrators to "abbreviate the verb, object, or qualifier(s)" of most commands "by typing the first few letters of each word, in the correct spelling order." *Id.* at 1-7. The documentation also explains that "[y]ou can build a command one word at a time. For example, at the beginning of the command line you can press F6 (Choices) to see a list of all valid command verbs. If you then type **ch** and press RETURN, the system expands **ch** to **change** and lists all valid words that can follow **change**. If you then type **sy** and press F6 (Choices) the system expands the command line to **change system-parameters** and lists all valid words that can follow." *Id.* This features allowed the administrator to "access all possible commands without any prior knowledge of specific commands, command components, or even the command structure." *Id.*

145. Table 2-1 of the Audix Screens documentation shows a complete list of commands available in the Definity Audix system, including indications of which command objects may follow a given command verb, and which command qualifiers may follow a given command object. *See id.* at 2-3–2-9. An excerpt from Table 2-1 appears below:



Verb	Object	Qualifiers
change	announcement	*announcement ID*
	auto-attend-routing	business-schedule holiday-schedule routing-table
	cos	*name or number*
	extensions	
	machine	*[machine name]*
	network-group	
	password	
	remote-subscriber	*name* I *address* I *machine-name extension*
	subscriber	*name* or *extension*
	switch link	
	switch time-zone	
	system-parameters	activity-log
		analog-network
		customer-options
		features
		imapi-options
		limits
		link-log (CL mode)
		maintenance
		outcalling
		password
		sending-restrictions
		thresholds
	transfer-dialplan	
	voice-group	
copy	annc-set	
	announcement	
	fragment	
disable	alarm origination	

Id. at 2-4.

146. If rotated 90 degrees clockwise, Table 2-1 takes on a more familiar tree form, with the highest level commands at the top, permissible objects below those commands, and permissible qualifiers below the objects. (There is an implied root node that is the parent of the verbs.)

147. Among the commands offered by Definity Audix was "test lan," which allowed an administrator to test the connectivity of a particular host on the network using the Definity Audix command line. Audix Screens at 3-71. An administrator could type "test lan dest *address*" (where address is an IP address) into the Definity Audix command line, and upon pressing enter, Definity Audix would "[p]erform[] the UNIX ping command which attempts to send a packet over the LAN to the host." *Id.* In other words, entering the Definity Audix command "test lan dest *address*" resulted in a *UNIX* command "ping" being executed for that destination address.

148. If an entered command was incomplete or used an invalid abbreviation, the Definity Audix system included the ability to notify the user of the error and to prompt the user for the missing information (e.g., a subscriber name, extension, or port number). *Id.* at 1-8.

1. Lucent Avaya Definity Audix System Release 4.0 Anticipates Claims 1 and 14

[1A] A method in a processor-based system configured for executing a plurality of management programs according to respective command formats, the method comprising:

149. In my opinion, Definity Audix practiced the preamble of claim 1.

150. Definity Audix was a processor-based voice messaging system. Its hardware included both a CPU, which was the "Multifunction board's main processor that controls system data transfer, input/output (I/O), and logical instructions," Audix Reference at GL-3, and a

"TN568 circuit pack," which included a digital signal processor that provided "signaling, power-level control, speech coding, and data processing." Audix Screens at GL-4.

151. The Definity Audix system practiced executing a plurality of management programs according to respective command formats. Definity Audix included a command line interface that allowed administrators to enter commands for configuring or administering the system. These commands could be used to access various "screens," which allowed the administrator to perform management tasks, such as adding subscribers to the system ("add subscriber"), changing a subscriber password ("change password"), or displaying configuration information ("list configuration"). Audix Screens at 1-8, 3-44, 3-151.

152. In some instances, entering a Definity Audix command resulted in executing a management program external to the Definity Audix system. For example, typing the command "test lan dest *address*" (where address would be the IP address of a host on the network) "[p]erforms the UNIX ping command which attempts to send a packet over the LAN to the host." Audix Screens at 3-71. One of ordinary skill in the art would understand that "ping" is a UNIX command, and that by entering the command "test lan dest *address*," the Definity Audix system, in turn, executed the ping command according to its respective format. In another example, typing "print display system-parameters features" caused a print job to be sent to the system printer connected to the user terminal. Getting Started at 22. One of ordinary skill in the art would recognize that "print display system-parameters features" is not a command that a printer would understand; using such a command to print would require Definity Audix to translate this command into a printer-specific command format.

153. Accordingly, it is my opinion that Definity Audix practiced "[a] method in a processor-based system configured for executing a plurality of management programs according

to respective command formats," under either party's proposed construction of "management programs."

[1B] receiving a generic command from the user;

154. In my opinion, Definity Audix practiced this limitation of claim 1 under either party's construction of the phrase "generic command."

155. Administrators (i.e., users) of Definity Audix accessed the system software via a terminal. *See, e.g.*, Audix Admin at 3-1. The terminal allowed these users to enter commands that would be received by the Definity Audix system. *Id.*; *see also* Audix Screens at 1-3–1-6. For example, to test the connectivity of a particular network host with the Definity Audix system, a user could enter "test lan dest *address*," as discussed above. Audix Screens at 3-71; *see also* Audix admin 7-6–7-7. As the Definity Audix documentation explains, "test lan dest *address*" command results in the system performing a UNIX "ping" command of the given host address. Audix Screens at 3-71. Because "test lan dest *address*" is a command that is an abstraction of the tool-specific UNIX "ping" command, which, as one of ordinary skill in the art would understand, has its own format and syntax, the Definity Audix "test lan dest *address*" command is a "generic command." Thus, upon a user's entering the "test lan dest *address*" command into the terminal, the Definity Audix system receives a generic command from that user.

156. By contrast, if a user types "test *alarm origination*," Definity Audix will "display whether alarm-origination is active," and not use the Unix "ping" command to do so, since that Unix command has an entirely different purpose. Audix Screens at 2-23, 3-11. Therefore, the generic command "test" is an abstraction of the command formats and syntaxes of more one management program.

157. Accordingly, it is my opinion that Definity Audix practiced "receiving a generic command from the user."

> *[1C] validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format, the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating step including identifying one of the elements as a best match relative to the generic command; and*

158. I understand that Cisco's proposed construction of "command parse tree" requires, in part, "a hierarchical data representation." *See* Ex. C, at 4. In my opinion, Definity Audix included a hierarchical data representation, as I explain below. In addition, I understand that Cisco's proposed construction of "command parse tree" requires "having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value." *See id.* To the extent Cisco reads this limitation as *not* requiring every single element of the tree to have a corresponding at least one command action value, it is my opinion that Definity Audix practiced this claim limitation for the reasons that follow. Additionally, it is my opinion that, at least under Arista's proposed construction of "command parse tree," Definity Audix practiced this claim limitation.

159. Definity Audix's command line interface included the ability to validate the generic command based on what the '526 patent calls a "command parse tree." As described above, Definity Audix commands were structured using a "verb-object syntax," where the verb specified the "type of action" to perform. Audix Screens at 1-5–1-6. Definity Audix allowed at least twenty such command verbs, and the system accepted truncated or abbreviated portions of unique command words, which allowed users to enter partial commands, such as "ch" instead of

"change." *Id.* at 1-6–1-7. The system also allowed users to type the F6 key to "see a list of all valid command verbs." *Id.* at 1-7. As the Audix Screens documentation explains:

> You can build a command one word at a time. For example, at the beginning of the command line you can press F6 (Choices) to see a list of all valid command verbs. If you then type **ch** and press RETURN, the system expands **ch** to **change** and lists all valid words that can follow **change**. If you then type **sy** and press F6 (Choices) the system expands the command line to **change system-parameters** and lists all valid words that can follow.

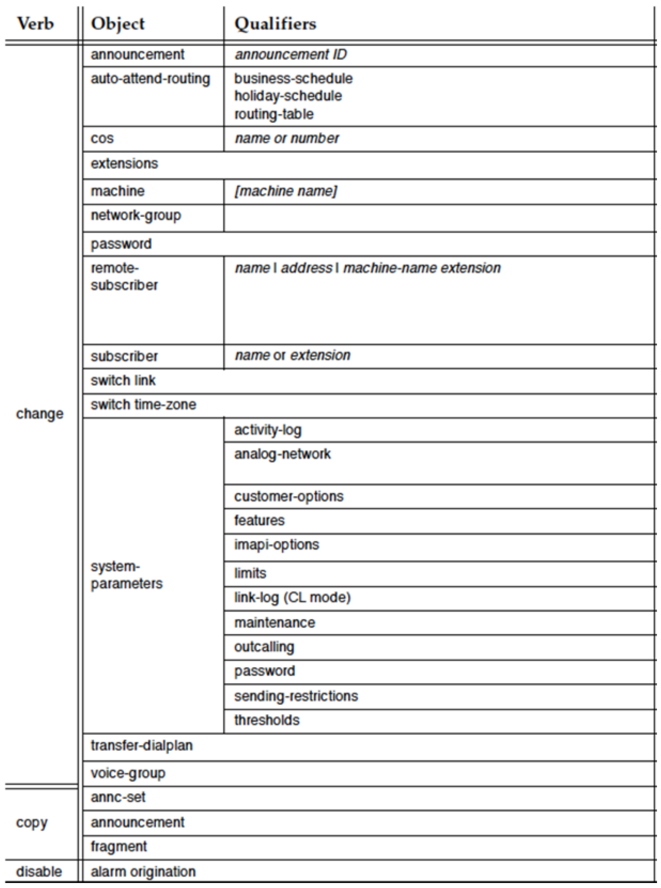
Id.

160. This abbreviation and help feature, in my opinion, discloses "validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format" under either party's proposed construction. Partially entered commands are validated when they are sufficiently unique that only one of the twenty command verbs can result from the truncated command entered. For example, when the user types "ch," the only possible valid command verb accepted by the system is "change." *See id.* at 1-6. The system thus validates the partial command relative to a prescribed generic command format.

161. The excerpt above further demonstrates that this validation occurs based on what the '526 patent calls a "command parse tree." As the example explains, a user can "build a command one word at a time," pressing the F6 command after each command word to receive a list of "all valid command words that can follow." *Id.* at 1-7. After entering "change," for example, pressing F6 will provide the user with a list of only those command objects that may permissibly follow the command verb "change." *Id.* The process can be repeated to determine which qualifiers may permissibly follow "change system-parameters." *Id.* Because Definity Audix included the ability to display only command components that could permissibly follow

previously entered command components (e.g., "system-preferences" after "change"), one of ordinary skill in the art would understand that this discloses a hierarchical data representation of the Definity Audix command structure.

162. Indeed, this understanding is confirmed by Table 2-1 of the Audix Screens documentation, which discloses this command structure for all potential commands in the Definity Audix system. *See id.* at 2-3–2-9. An excerpt of Table 2-1 appears below:

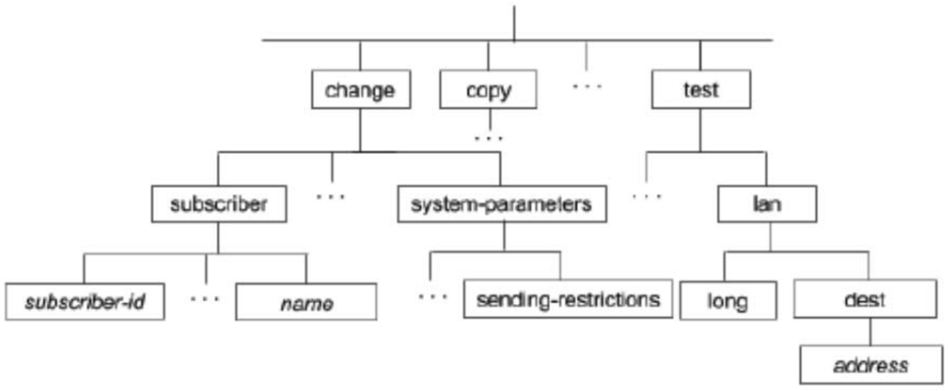


Verb	Object	Qualifiers
change	announcement	*announcement ID*
	auto-attend-routing	business-schedule holiday-schedule routing-table
	cos	*name or number*
	extensions	
	machine	*[machine name]*
	network-group	
	password	
	remote-subscriber	*name* | *address* | *machine-name extension*
	subscriber	*name* or *extension*
	switch link	
	switch time-zone	
	system-parameters	activity-log
		analog-network
		customer-options
		features
		imapi-options
		limits
		link-log (CL mode)
		maintenance
		outcalling
		password
		sending-restrictions
		thresholds
	transfer-dialplan	
	voice-group	
copy	annc-set	
	announcement	
	fragment	
disable	alarm origination	

Id. at 2-4.

163. The command parse tree represented in Table 2-1 can be represented in a more familiar tree form if simply rotated 90 degrees clockwise and, as I said above, have its implied

root node added at the top. I have prepared the figure below, which excerpts exemplary commands from Table 2-1, to illustrate this. This figure includes nodes and branches for the "change" and "test" commands from Table 2-1. If presented with all commands in the Defınty Audix system, the top level of the command parse tree below would include the twenty valid Definity Audix command verbs, as discussed above. *See id.* at 1-6. The branch below each command verb would include each command object that could permissibly follow the command verb. For purposes of this illustration, I have focused on two branches of command objects that could permissibly follow the "change" command ("subscriber" and "system-parameters") and one branch that could follow the "test" command ("lan"). *See id.* at 2-4, 2-9; *see also id.* at 1-6, 1-7. Beneath each of these are additional qualifiers that can permissibly follow. *See id.* at 1-7 ("change system-parameters sending-restrictions" and "change subscriber *subscriber-id*"), 3-164 ("change subscriber *name*"), 2-3. Each of these command verbs, command objects, and command qualifiers are what the '526 patent refers to as "command components."



164. Definity Audix further discloses that associated with each command component of the command parse tree is an action that occurs when that component is the final valid valid component of a user-entered generic command. The Audix Screens documentation explains, for

example, the various possible command actions associated with the "test LAN" command. If the user entered the command "test lan," it would result in Definity Audix performing a "short test that checks the basic operation of the LAN interface software and hardware . . . ." *Id.* at 3-71. Adding the component "long"—"test lan long"—would "[p]erform a long test that tests the basic operation of LAN interface software and hardware . . . ." *Id.* Finally, as discussed above, typing "test lan dest *address*" would "[p]erform the UNIX ping command which attempts to send a packet over the LAN to the host." *Id.*; *see also id.* at Table 2-1 (displaying valid command components of "test").

165. In validating the user-entered generic commands, the Definity Audix system identified one of the elements of the command parse tree as a "best match" relative to the generic command. As discussed above, Definity Audix included a command line abbreviation feature that allowed a user to enter a partial or truncated command. If the portion of partial command entered was sufficiently unique, Definity Audix would identify the best match for that command relative to the generic command. For example, a user could enter "ch sy s" in the command line, and Definity Audix would recognize this command as "change system-parameters sending-restrictions." *Id.* at 1-7. In this example, the final element "sending-restrictions" would be the "best match" relative to the entered generic command "ch sy s" because it is both the last entered and last valid component of the command. Similarly, for the generic command "test lan dest *address*," the command component "*address*" would be the "best match" for that generic command, provided "address" was in the form of a valid IP address.

[1D] issuing a prescribed command of a selected one of the management programs according to the corresponding command format, based on the identified one element.

166. In my opinion, Definity Audix practiced this limitation of claim 1 under either party's proposed construction of "management programs."

167. As discussed with respect to the previous limitation, when a user entered the generic command "test lan dest *address*" (where address is a host IP address), Definity Audix would "[p]erform the UNIX ping command which attempts to send a packet over the LAN to the host." *Id.* at 3-71. One of ordinary skill in the art would understand that "ping" is a native shell command for the UNIX environment that uses a different format and syntax than "test lan dest *address*." The UNIX ping command is the "prescribed command" that results from the user entering the generic command "test lan dest *address*" into the Definity Audix terminal. As discussed above, the ping command is issued only if the user enters the valid generic command "test lan dest *address*," where "address" is in the form of an IP address. Because a valid address is the final element of the generic command, once validated, the ping command that issues is thus based on that identified element of the command parse tree. Similarly, had a user entered "test lan long," the final element "long" would indicate to Definity Audix to issue a different prescribed command (e.g., performing the long version of the LAN test).

168. Accordingly, it is my opinion that Definity Audix practiced "issuing a prescribed command of a selected one of the management programs according to the corresponding command format, based on the identified one element."

Claim 14

[14A] A computer readable medium having stored thereon sequences of instructions for executing a plurality of management programs according to respective command formats, the sequences of instructions including instructions for performing the steps of:

[14B] receiving a generic command from the user;

> *[14C] validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format, the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating step including identifying one of the elements as a best match relative to the generic command; and*
>
> *[14D] issuing a prescribed command of a selected one of the management programs according to the corresponding command format, based on the identified one element.*

169. Claim 14 is substantially identical to claim 1, which I address above, except that it is drafted as a computer readable medium storing instructions for performing the method claimed in claim 1. One of ordinary skill in the art would understand from the Audix Reference and the Audix Admin, which describe aspects of the operation of Definity Audix, that Definity Audix consisted of instructions for performing the claimed method and that such instructions were stored on a computer readable medium, such as RAM, ROM, and/or a disc. Accordingly, for the same reasons set forth as to claim 1, it is my opinion that Definity Audix anticipates claim 14.

[redacted]

[redacted]

[redacted]

ATTORNEYS' EYES ONLY

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[11] ARISTANDCA000002349

[redacted]

[redacted]

[redacted]

[redacted]

[redacted] [redacted]

[redacted]

[redacted]

[redacted]

[redacted] [redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted] [redacted]

[redacted]

[redacted]

[redacted]

[redacted] [redacted]

[redacted]

[redacted] [redacted]

[redacted]

[redacted] [redacted]

[redacted]

[redacted]

[redacted] [redacted]

[redacted]

[redacted] [redacted]

[redacted]

[redacted]

[redacted]

[redacted] [redacted]

[redacted] [redacted]

[redacted]

[redacted] [redacted]

[redacted]

[redacted]

[redacted] [redacted]

[redacted]

[redacted]

[redacted] [redacted]

[redacted] [redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted] [redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted] [redacted]

[redacted]

[redacted].

D. **Martinez-Guerra**

192. U.S. Patent No. 6,523,172 to Martinez-Guerra, et al. ("Martinez-Guerra")[12] is entitled "Parser Translator System and Method." I understand that Martinez-Guerra is prior art to the '526 patent under 35 U.S.C. § 102(e) as it was filed on February 19, 1999, claiming priority to a December 1998 provisional application, and issued on February 18, 2003.

193. Martinez-Guerra discloses a parser-translator that allows a user to input a command written in a "high-level user language" and then translates that command into "logically and syntactically correct directives" for specific software tools. Martinez-Guerra, Abstract. As in the '526 patent, a benefit of the disclosed parser-translator is that "a user can focus on the semantics of the desired operation and need not be concerned with the proper syntax of language for a particular system." *See id.* Abstract, 10:7-11, 13:30-33.

194. The parser-translator disclosed by Martinez-Guerra includes a token recognizer, a parser, and a translator. *Id.* at 9:24-26. Martinez-Guerra discloses various implementations of the parser-translator, which utilize a data structure representing a language's grammar in the course of performing a translation. *See, e.g.*, 5:51-58, 6:29-44, 18:31-35, Figs. 1-3 (showing "Grammar 17," "Grammar 27," and "Grammar 37"). Although variations in the context and capabilities of external data handling are disclosed, the essential token processing, parsing, and translation functions taught are consistent across Martinez-Guerra's embodiments. *Id.* at 9:6-10 ("FIG. 1 depicts an exemplary data conversion system employing a parser-translator to at least partially define a specific source-to-target data conversion in accordance with a grammar

[12] ARISTANDCA00008869

encoding legal statements in a data transformation language."); *id.* at 11:48-51 ("FIG. 3 depicts an illustrative parser-translator component 30 suitable for use in a variety of software tool environments to parse a stream of tokens in accordance with a grammar . . . ."). Figure 1 of Martinez-Guerra shows how the parser-translator is structured:

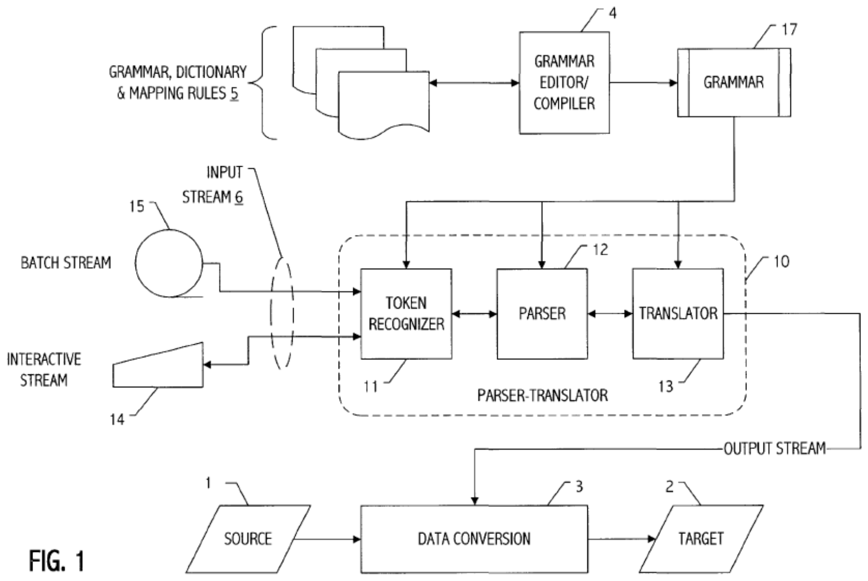


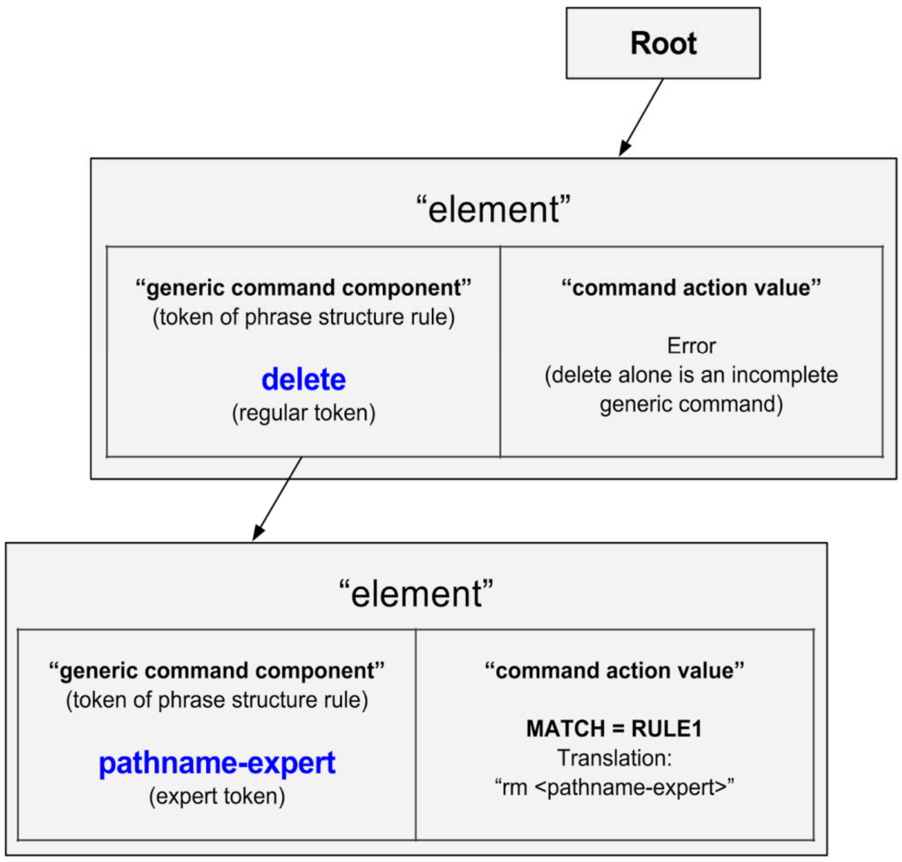
FIG. 1

195. The disclosed parser-translator receives an input "from which a token recognizer 11 extracts tokens in accordance with dictionary entries and rules encoded in grammar 17 and in accordance with a current parse state." *Id.* at 9:36-40. (I note that Martinez-Guerra uses the term "tokens" to describe the input received from the user, while the '526 patent refers to such input as "command words.") Specifically, the parser receives generic commands, such as "delete /usr/extract/testing," from a user via an input stream. *Id.* at 9:35-36, 15:50-62. The translator translates those generic statements into tool-specific outputs based on rules specified in the "grammar." *Id.* at 9:42-44, 3:29-37. The grammar defines valid sequences of words from the input stream through a set of "phrase structure rules," and corresponding translation rules that tell the parser-translator how a valid input statement should be translated once a phrase structure

rule is satisfied. *Id.* at 9:40-45, 7:52-64. Martinez-Guerra points out that "a variety of suitable representations and encodings of a grammar are suitable . . . .[P]ersons of ordinary skill in the art will appreciate a variety of alternative representations and encodings." *Id.* at 13:59-65. A POSA would understand that a tree structure is one such suitable representation, if the language described by the grammar is finite (has only a finite number of possible valid statements). A POSA would also understand that finite command languages constitute a subset of the languages addressed by Martinez-Guerra, which describes translating commands for a variety of tools, just as a POSA would understand that the command languages addressed by the '526 patent are finite. The grammar also uses dictionary entries that contain all of the input words that the parser-translator should recognize. *Id.* at 14:16-18. Dictionary entries correlate input words with grammar categories. *Id.* at 7:36-37. Some tokens in the dictionary are called "expert tokens"—they denote a grammar category of acceptable user inputs rather than a specific one. A filename path would be an example of such a category; an even number would be another. *Id.* at 7:38-44, 15:6-12.

196. As I explain in greater detail below, for any finite command language, Martinez-Guerra's phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid sequences of components of statements in a high-level language, and because every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed. The phrase structure rule RULE1, which I use as an example, would be represented as a set of component-action value pairs (what the '526 patent calls "elements") within such a command parse tree, as I have illustrated in the figure below:

PHRASE STRUCTURE RULE: RULE1
delete pathname-expert



197. As the input stream is processed, the parser-translator uses the phrase structure rules, which, as illustrated in the figure above, constitute a command parse tree, to determine valid ("legal") next parse states—i.e., the tokens that, as defined by the grammar, may follow the token(s) already received. *Id.* at 10:42-58, 18:31-48. The parser "builds an internal representation of the translation of the input stream according to translation rules encoded in the grammar." *Id.* at 10:50-53. When a valid sequence of received tokens uniquely matches a single phrase structure rule, the parser-translator can carry out the translation corresponding to that rule. *Id.* at 10:56-58, 20:45-49. Once translated, the output statement (e.g., a prescribed command) for a given software tool is issued and can thereafter be executed. *See, e.g.*, id. at 15:50-62.

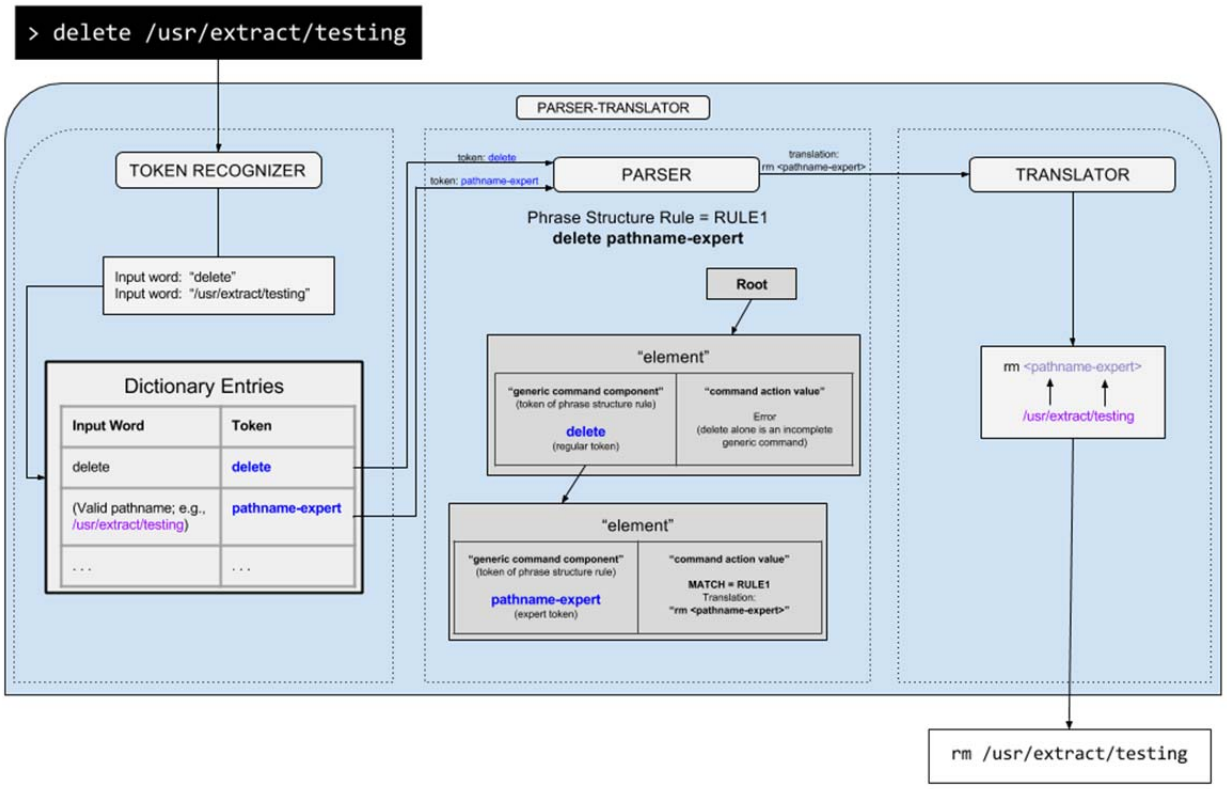
198. Throughout my analysis of the '526 patent claims, I use Martinez-Guerra's discussion at column 15:50-16:5 as a working example of how its teachings disclose the relevant limitations. This discussion explains how the parser-translator would translate a generic command received from a user ("delete /usr/extract/testing") into the corresponding prescribed command ("rm /usr/extract/testing") using the phrase structure rule "RULE1." RULE1 describes the legal ordering of tokens ("delete pathname-expert") that will satisfy the rule. Once the rule is satisfied, the parser-translator will perform the translation from the generic command to the prescribed command. *Id.* at 8:24-26, 14:10-11, 15:54-59. The tokens that are compared to those in RULE1 are determined by matching each input word of the generic command received from the user ("delete /usr/extract/testing") to a corresponding token found in the dictionary entries of the grammar. *Id.* at 9:34-38, 14:60-15:15. Martinez-Guerra explains that the dictionary entries identify all of the tokens that the parser-translator should recognize, and acts as a table for correlating an input command word with a corresponding token (here I use "token" in the sense of the '526 patent). *Id.* at 14:16-18, 14:60-62, 20:22-24.

199. In this example, the generic command includes two command words, "delete" and "/usr/extract/testing." Before the command words are compared with RULE1, each word is checked to confirm that it is present in the dictionary entries, which confirm that the input words are valid for use with the parser-translator. *Id.* at 9:34-38, 15:54-16:5. "[D]elete" is a "regular token" because it is a fixed sequence of characters, and the content of the token would be the same as the input word ("delete"). *Id.* at 15:2-4, 15:58-59. The input word "/usr/extract/testing" is what Martinez-Guerra calls an "expert token," which is "a string whose value cannot be specified when the grammar is written but which can be defined in terms of its characteristics." *Id.* at 15:4-6, 15:67-16:5. The expert token "receives its value from the user, and the grammar

imposes restrictions on the type of value accepted." *Id.* at 15:6-8. The input word "/usr/extract/testing" specifies the pathname of a particular file that the user wishes to operate upon. Because there are countless files that a user could identify for this command word, Martinez-Guerra explains that the dictionary entries include validation functions for determining that the word "/usr/extract/testing" is a pathname, as opposed to a regular token. This input word corresponds to the token "pathname-expert." *Id.* at 15:6-15, 17:10-24, 20:10-28. The parser-translator therefore matches "/usr/extract/testing" with the token "pathname-expert" in the dictionary entries. Thereafter, when determining the presence of tokens in elements of the phrase structure rules, the parser-translator will look for the token "pathname-expert" in a rule in the place of the input word "/usr/extract/testing." *Id.* at 15:54-16:5.

200. If the sequence of tokens matches a sequence in a phrase structure rule, then that rule is satisfied, and a corresponding translation occurs. *Id.* at 9:40-43, 20:45-48. Thus, because the tokens "delete" and "pathname-expert" satisfy RULE1, the translator is instructed to perform the translation as "rm <pathname-expert>." *Id.* at 15:59-62. Martinez-Guerra explains that, in this example, "<pathname-expert>" will be replaced with the original input word, thus resulting in the translator issuing the prescribed command "rm /usr/extract/testing." *Id.* at 15:65-16:5.

201. I have prepared the following graphic which illustrates the working example from column 15:50-16:5, as discussed above.


> delete /usr/extract/testing
PARSER-TRANSLATOR
TOKEN RECOGNIZER
Input word: "delete"
Input word: "/usr/extract/testing"
Dictionary Entries
Input Word
Token
delete
delete
(Valid pathname; e.g., /usr/extract/testing)
pathname-expert
. . .
. . .
token: delete
token: pathname-expert
PARSER
translation: rm <pathname-expert>
Phrase Structure Rule = RULE1
delete pathname-expert
Root
"element"
"generic command component" (token of phrase structure rule)
delete (regular token)
"command action value"
Error (delete alone is an incomplete generic command)
"element"
"generic command component" (token of phrase structure rule)
pathname-expert (expert token)
"command action value"
MATCH = RULE1 Translation: "rm <pathname-expert>"
TRANSLATOR
rm <pathname-expert>
/usr/extract/testing
rm /usr/extract/testing

202. Martinez-Guerra teaches that, as each word in a command is parsed, the phrase structure rules define a set of valid next tokens in light of the tokens that have already been validated. *See, e.g.*, *id.* at 18:36-40 ("Token recognizer 31 computes a set of valid next states . . . [that] includes the set of next tokens consistent with a current parse state.") At first, the set of valid tokens includes all "tokens that may begin a legal statement in the language defined by the grammar." *Id.* at 18:40-42. Each successive token is then compared to the set of valid next tokens in light of the tokens that came before. *See, e.g.*, *id.* at 20:31-34 ("Token recognizer 31 performs input matching against each of the valid next states corresponding to a parse state representation consistent with the input stream read so far.")

1. Petition for *Inter Partes* Review

203. I understand that on May 17, 2016, the Patent Trial and Appeal Board ("PTAB") of the U.S. Patent and Trademark Office issued a decision denying institution of *inter partes* review of the asserted '526 patent claims based on Martinez-Guerra. Ex. ___. I wrote a declaration in that matter, have reviewed the PTAB decision, and have considered it in my opinion expressed here regarding Martinez-Guerra.

204. In summary, I understand that the PTAB denied institution because it determined Martinez-Guerra lacks the claimed "command action value," which the PTAB construed to mean "a value that can be used to identify a prescribed command and is contained in each element of the parse tree." *Id.* at 12. Specifically, the PTAB determined that Martinez-Guerra's error reporting "is not a value used to identify a prescribed command, as required by our claim construction. The error does not indicate a prescribed command. Additionally, [the PTAB concluded], the error is not contained in each element of the parse tree; rather it is only at the particular branch of the tree that causes the error." *Id.* at 15.

205. In reaching this determination, the PTAB noted its reliance on the patent owner's (Cisco's) statement that "in contrast to the '526 patent, each token ingested by Martinez-Guerra's parser does not have a corresponding translation function. Instead, Martinez-Guerra's translation functions are only applied once Martinez-Guerra's system determines that it has received the entirety of a complete valid sequence of tokens." *Id.* at 16. Agreeing with this statement, the PTAB concluded that there was insufficient evidence that Martinez-Guerra included a "command action value at each element of the parse tree." *Id.*

206. I understand that the PTAB's decision and claim construction are not binding in this litigation.

2. Martinez-Guerra Anticipates the Asserted Claims

[1A] A method in a processor-based system configured for executing a plurality of management programs according to respective command formats, the method comprising:

207. In my opinion, the preamble language of claim 1 is satisfied by Martinez-Guerra.

208. Martinez-Guerra discloses a parser-translator that uses a high-level user language to interface with a plurality of software applications. Martinez-Guerra at 3:48-52. The parser-translator translates statements from a high-level language, which can include generic statements, to "logically and syntactically correct directives for performing the desired data transformations or operations," e.g., prescribed commands in the proper format, using one or more software tools. *Id.* at 3:29-37. The method disclosed by Martinez-Guerra is carried out in software on a computer system. *Id.* at 9:10-13.

209. In my opinion, Martinez-Guerra discloses "management programs" under either party's proposed construction of the term because it provides examples of software tools that both integrate the disclosed parser-translator and software tools that are separate from or external to it. Moreover, as with the management programs of the '526 patent, the exemplary software tools in Martinez-Guerra use their own respective formats to provide management functions.

210. Martinez-Guerra broadly describes its parser-translator, indicating a wide range of embodiments wherein the parser-translator is either internal or external to the disclosed software tools. It indicates that the parser-translator may be "integral with a software tool or provided separately as a component suitable for integration with a software tool or tools" or provided "as a standalone tool for guiding users through the definition of user language statements." *Id.* at 22:48-54. One of ordinary skill in the art would understand from these disclosures that Martinez-Guerra intended that the parser-translator could either be integrated into such a

software tool or used to provide translations to one or more software tools separate from or external to a standalone parser-translator.

211. For example, Martinez-Guerra explains that "illustrative enterprise tools environments includ[es] data discovery and cleansing tools, data extraction conversion and migration tools, data movement and replication tools, query Multi-Dimensional Data (MDD) analysis and On-Line Analytical Processing (OLAP) tools, as well as applications and gateways that may advantageously incorporate a parser-translator component." *Id.* at 11:36-42; *see also id.* at 13:17-20 (disclosing software applications such as "query and on-line analytical processing tools, gateways to operational data systems" and "enterprise resource planning applications"). It likewise indicates that "[i]n some embodiments, a single parser-translator component provides multiple software applications with an interface to high-level user language statements wherein operation of the single parser-translator component is suitably defined for each software application using a corresponding grammar coding." *Id.* at 3:48-53.

212. The disclosed software tools use a command format that differs from the high-level, generic command format provided by the user. *See id.* At 15:50-62. For example, Martinez-Guerra's example in column 15 illustrates that one generic command "delete /usr/extract/testing" is translated for a UNIX environment to the prescribed UNIX command "rm /usr/extract/testing." *Id.* Indeed, Martinez-Guerra states that a benefit of the disclosed parser-translator is that "a user can focus on the semantics of the desired operation and need not be concerned with the proper syntax of language for a particular system." *See id.* at Abstract, 10:7-11, 13:30-33.

213. Accordingly, it is my opinion that Martinez-Guerra discloses the preamble.

[1B] receiving a generic command from the user;

214. Martinez-Guerra discloses receiving from a user an input stream consisting of statements in a "high-level user language[.]" *Id.* at Abstract; *see also* Fig. 1, 9:15-23. It is my opinion that these statements constitute a "generic command" at least under at least Cisco's construction of the term "generic command.

215. As Martinez-Guerra explains, the system "allows users to build complex statements (e.g., using natural languages within a user interface environment) and to translate those complex statements into statements or directives appropriate to a particular data processing application." *Id.* at 4:4-9. By using these "high-level" generic commands, "the user can focus on the semantics of the desired transformations and need not be concerned with the proper syntax of query or populate statements for a particular data access system, access language or database installation." *Id.* at 10:7-11. In other words, these statements are abstractions of tool-specific commands that allow a user to input a command based on the relative function, as opposed to the specific syntax, for that tool.

216. For example, Martinez-Guerra discloses that the parser-translator may receive an input stream including the command "delete /usr/extract/testing." *Id.* at 15:50-16:5 ("the string delete is an input stream such as input stream 36"). This is a "generic command," as that term is used in the '526 patent, because it provides an abstraction of the UNIX tool-specific command "rm /usr/extract/testing." One of ordinary skill in the art would understand that "delete /usr/extract/testing" is analogous to the "Generic Command Examples" disclosed in the '526 patent. *See* '526 patent at 7:1-9:15 (identifying, e.g., "set watchtime <milliseconds>" as an exemplary generic command). As explained in the limitations that follow, the generic command "delete /usr/extract/testing" received from the user in the input stream is validated by the parser-

translator and ultimately translated into the prescribed UNIX command "rm /usr/extract/testing" according to a translation function encoded in the grammar.

217. Accordingly, it is my opinion that Martinez-Guerra discloses "receiving a generic command from the user" at least under Cisco's proposed construction of "generic command."

> *[1C] validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format, the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating step including identifying one of the elements as a best match relative to the generic command; and*

218. In my opinion, Martinez-Guerra discloses the above limitation of claim 1.

219. I understand that Cisco's proposed construction of "command parse tree" requires, in part, "a hierarchical data representation." *See* Ex. C, at 4. In my opinion, Martinez-Guerra included a hierarchical data representation, as I explain below. In addition, I understand that Cisco's proposed construction of "command parse tree" requires "having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value." *See id.* To the extent Cisco reads this limitation as not requiring every single element of the tree to have a corresponding at least one command action value, it is my opinion that Martinez-Guerra practiced this claim limitation for the reasons that follow. Additionally, it is my opinion that, at least under Arista's proposed construction of "command parse tree," Martinez-Guerra practiced this claim limitation.

220. Martinez-Guerra discloses that words in the input stream (which Martinez-Guerra refers to as "tokens") are analyzed by the token recognizer subcomponent of the parser-translator according to dictionary entries and the rules encoded in the grammar. *Id.* at 10:35-38. Martinez-Guerra defines a "grammar" as "[a] formal definition of the syntactic structure of a language

typically represented as a set of rules that specify legal orderings of constituents and subconstituents (e.g., phrases and symbols) in properly formed statements or sentences in a language." *Id.* at 7:52-56. The "legal orderings" refer to valid sequences of tokens as defined by phrase structure rules in the grammar. *Id.* at 8:24-26. The grammar also provides translation functions corresponding to all such valid sequences of tokens. *Id.* at 16:15-16, 17:51-52.

221. Martinez-Guerra explains that the parser-translator creates internal data structures, including "parse state data structures" and "internal representations of a grammar," which it uses to keep track of next legal choices in the grammar. *Id.* at 18:31-40, 20:31-34, and 10:42-49. "[T]he set of valid next states includes the set of next tokens consistent with the current parse state." *Id.* at 18:38-40. Stated differently, the parser-translator validates the words in the user input stream using the phrase structure rules, which define valid next tokens in light of the tokens that have already been validated. In the case of rules such as RULE1 (and those rules illustrated in Figure 2 of the '526 patent), phrase structure rules together constitute a command parse tree, which specifies the possible valid next legal choices until the input stream has been fully parsed. *Id.* at 10:42-58, 20:45-49. A person of ordinary skill in the art would understand that for finite command languages, Martinez-Guerra's phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid components of statements in a high-level (i.e., "generic") command language, and because, as I demonstrate below, every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed.

222. When the input stream contains a generic command, as in the "delete" example (*id.* at 15:50-62), the command is validated by the parser-translator relative to a prescribed generic command format defined by phrase structure rules in the grammar. *Id.* at 10:42-58.

Specifically, the token recognizer subcomponent receives the "delete" command and first determines whether the word "delete" matches a token in the dictionary entries, which define all tokens found in the phrase structure rules of the grammar. *Id.* at 9:34-38, 14:16-18. Once it has confirmed a match, the token recognizer then determines whether the token (in the '526 sense) corresponding to "delete" is present among the next legal parse states maintained by the parser. *Id.* at 9:34-40, 20:28-34.

223. As discussed above, each parse state corresponds to the parser's arrival at a node in the command parse tree, which shows all valid next legal tokens, as defined by the phrase structure rules. In the example found in column 15, the tokens "delete" and "pathname-expert" are matched, respectively, with the first and second words of the phrase structure rule, "RULE1." *Id.* at 15:50-62. Because a phrase with the tokens "delete" and "pathname-expert," in that order, matches the phrase structure rule RULE1—i.e., the "prescribed generic command format"—the generic command is validated.

224. As discussed above, Martinez-Guerra discloses that the grammar includes "a set of phrase structure rules and associated translation rules that define the syntactic order of a source and target language . . . .". *Id.* at 7:57-59, 17:51-52. A person of ordinary skill in the art would understand that for finite command languages, Martinez-Guerra's phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid components of statements in a high-level (e.g., "generic") command language, and because, as I discuss below, every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed. Phrase structure rules like "RULE1" together define legal orderings of tokens according to a particular language, such as a high-level language (*id.* at 14:11-12), and can be represented as a set of textual rule specifications (e.g.,

"RULE1 → delete pathname-expert"). In the textual rule specification for RULE1, each token to the right of the arrow is a component needed to form a complete statement; each of these is what the '526 patent calls a "generic command component."

225. Corresponding to each of these components of the phrase structure rule is an appropriate action that will occur if there are no further words to validate against the command parse tree (if that component is a "best match" for the generic command). If an input word is invalid or a command is incomplete—for example, if only the word "delete" is received without any pathname to define the target of the operation—that action may result in returning an error. *See id.* at 19:3-4. On the other hand, a valid input stream that matches and completes a phrase structure rule—e.g., "delete /usr/extract/testing" (*id.* at 16:5)—will correspond to performing the translation associated with the phrase structure rule RULE1. *Id.* at 8:60-65. The instruction to return an error that results from inputting an invalid word or an incomplete sequence (e.g., inputting "delete" without a target pathname), or the translation function that results from inputting a complete and valid sequence of tokens (e.g., "delete /usr/extract/testing"), is what the '526 patent refers to as a "command action value." In Martinez-Guerra, such a command action value exists for each generic command component of a given phrase structure rule because, as input words are processed, they may be invalid, resulting in an error, or valid, leading ultimately to the satisfaction of a phrase structure rule, thus resulting in a translation according to the corresponding translation function.

226. Thus, in Martinez-Guerra, each legal choice according to a phrase structure rule is an "element," as described in the '526 patent, which has both a generic command component and a corresponding command action value.

227. In the generic command example discussed above, Martinez-Guerra explains that the generic command "delete /usr/extract/testing" is validated by the parser-translator because it matches the legal ordering of tokens described by the phrase structure rule RULE1. *Id.* at 15:50-62. The element corresponding to this final token of RULE1, "pathname-expert," is the "best match" relative to the generic command because it is the final thing necessary to satisfy the phrase structure rule, thus causing the parser-translator to translate the input stream into a prescribed command. As Martinez-Guerra states, "[o]nce the current parse state includes a complete phrase as defined by phrase structure rules encoded in grammar 37, translator 33 provides a translation." *Id.* at 11:57-60.

228. Accordingly, it is my opinion that Martinez-Guerra discloses "validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format, the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating step including identifying one of the elements as a best match relative to the generic command."

[1D] issuing a prescribed command of a selected one of the management programs according to the corresponding command format, based on the identified one element.

229. Martinez-Guerra discloses that when a phrase structure rule is satisfied by the input stream, the translation function associated with the phrase structure rule causes the parser-translator to perform the translation. *Id.* at 11:55-62, 14:10-21, 17:47-52. The translator outputs the translation in a format for a software tool. *Id.* at 13:29-33. As Martinez-Guerra explains, "the translation functionality of parser-translator component 30 provides the capability of

translating the now properly formed user language statements to the form and syntax desired by a specific software tool." *Id.*

230. In the case of the generic "delete /usr/extract/testing" command, after matching the final token with a next legal choice in the parse state, Martinez-Guerra explains that the parser-translator carries out the translation using the translation function associated with phrase structure rule "RULE1," resulting in the prescribed UNIX command "rm /usr/extract/testing." *Id.* at 15:59-62, 17:51-52. One of ordinary skill in the art would recognize that "rm" is the UNIX shell command used to delete files in a computer system. Martinez-Guerra thus discloses issuing a prescribed command for removing a file in a selected one of the management programs according to the corresponding command format for that program ("rm /usr/extract/testing"). *Id.* at 15:50-62, 16:5.

231. Accordingly, it is my opinion that Martinez-Guerra discloses "issuing a prescribed command of a selected one of the management programs according to the corresponding command format, based on the identified one element."

a. Non-Asserted Claim 2

232. I understand that claim 2 is not currently asserted against Arista, but claim 6, which depends from claims 2-5, is asserted. Because I understand that the limitations of claim 2 are thus part of claim 6, I will address it below.

[2A] The method of claim 1, wherein the generic command includes at least one input command word, the validating step including:

233. In my opinion, Martinez-Guerra discloses this limitation of claim 2.

234. Martinez-Guerra discloses a generic command "delete /usr/extract/testing," which includes at least one input command word, "delete." *Id.* at 15:50-62. As explained, "delete" in RULE1 is a token "matching the string delete in the input stream such as input stream 36." *Id.*

235. Accordingly, it is my opinion that Martinez-Guerra discloses that "the generic command includes at least one input command word."

[2B] comparing each input command word to a command word translation table, configured for storing for each prescribed command word a corresponding token, for identification of a matching token; and

236. In my opinion, Martinez-Guerra discloses this limitation of claim 2.

237. The token recognizer subcomponent of the parser-translator extracts input command words from the input stream "in accordance with dictionary entries and rules encoded in grammar 17 and in accordance with a current parse state." *Id.* at 9:35-38, 11:51-55. Martinez-Guerra explains that dictionary entries "encode information associated with tokens" (*id.* at 10:24-25), and "describe all the tokens that a parser-translator (e.g., token recognizer 31 of parser-translator component 30) should recognize . . . ." *Id.* at 14:16-18. The '526 patent describes the "command word translation table" in similar terms: it "includes all the command words 26 that are valid according to the generic syntax." '526 patent at 3:44-46. One of ordinary skill in the art would understand from these disclosures, and in view of Martinez-Guerra's example of the "delete /usr/extract/testing" generic command, that the dictionary entries in Martinez-Guerra are a translation table used by the token recognizer to match input words with valid tokens found in the phrase structure rules (e.g., "delete" in the exemplary "RULE1"). *See* Martinez-Guerra at 14:55-59. As Martinez-Guerra explains, "[a] rule specification includes a sequence of grammar categories that define a legal sequence (or sequences) of tokens in the language. Tokens (read from the input stream) 'match' categories." *Id.*

238. For example, when the user submits the "delete /usr/extract/testing" command, the first input word "delete" is matched with a "regular token" entry in the dictionary having the same content ("delete"). As Martinez-Guerra explains, a "regular token is a fixed sequence of characters (for example, the string 'delete')." *Id.* at 15:2-4. The next (and final) input word the user submits in this example is the pathname for the file to be deleted ("/usr/extract/testing"). *Id.* at 15:57-16:5. This input word is what Martinez-Guerra refers to as an "expert token," and the parser-translator uses the dictionary entries to match that input word with the token "pathname-expert." *Id.* at 15:4-15, 15:63-16:5. The parser-translator then uses the matching token "pathname-expert" from the dictionary entries instead of the original input word "/usr/extract/testing" when searching for a match in the command parse tree.

239. Accordingly, it is my opinion that Martinez-Guerra discloses "comparing each input command word to a command word translation table, configured for storing for each prescribed command word a corresponding token, for identification of a matching token."

[2C] determining a presence of the matching token within the command parse tree for each input command word.

240. For each input command word that the token recognizer subcomponent matches with a corresponding token in the dictionary (id. at 11:51-55), it determines whether that token is among the next legal choices in the parse states maintained by the parser. *Id.* at 18:36-42, 18:66-19:7, 20:31-34, and 10:42-49. A person of ordinary skill in the art would understand that for any finite command language, Martinez-Guerra's phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid components of statements in a high-level command language, and because, as I discussed above, every component of each phrase structure rule has an action that will occur if that component is the last

valid component parsed. *See* supra at ¶ ___; *see also id.* at 14:10-15, 18:38-48. If the parser-translator determines that an input token matches an expected next legal choice in the parse state, then it adds the token to the parse state, and the parser computes an updated list of potentially valid next states. *Id.* at 19:5-35. Stated in terms of the '526 patent, the parser-translator determines the presence of a matching token (e.g., "delete") within the command parse tree.

241. In the example discussed in the previous limitation, the parser-translator first determines the presence of the token corresponding to the input word "delete" in the command parse tree. *Id.* at 15:50-62, 18:31-48. A matching token is found within the command parse tree because "delete" is the first token in the phrase structure rule RULE1. *Id.* at 15:57; 20:40-45, Fig. 10. Having matched the first token, the parser-translator then determines the presence of the second (and final) token, which Martinez-Guerra explains is the token "pathname-expert," corresponding to a valid file path name supplied by the user ("/usr/extract/testing"), as explained in the previous limitation. *Id.* at 15:50-16:5, 20:40-45. The presence of "pathname-expert" would also be found in the command parse tree because it corresponds to the final token of the phrase structure rule RULE1. *Id.* at 15:57. The parser-translator of Martinez-Guerra thus determines the presence of a matching token in the command parse tree for each input command word.

242. Accordingly, it is my opinion that Martinez-Guerra discloses "determining a presence of the matching token within the command parse tree for each input command word."

b. Non-Asserted Claim 3

243. I understand that claim 3 is not currently asserted against Arista, but claim 6, which depends from claims 2-5, is asserted. Because I understand that the limitations of claim 3 are thus part of claim 6, I will address it below.

[3] The method of claim 2, wherein the determining step includes recursively traversing the command parse tree based on an order of the input command words for identification of the matching token within the identified one element.

244. I come now to the phrase "recursively traversing." I agree with Cisco's expert Dr. Kevin C. Almeroth, who opined on this in his declaration in support of Cisco's claim construction brief: " 'Recursively traversing [a]…tree' is a concept that is well known in the art[,]" and further opined that "[i]n the art, traversing a tree is a high-level concept that describes ***systematically processing each element of the tree***." (emphasis added), Almeroth Decl., paragraphs 71, 75. His explanation of "recursively" was less clear, in part because he relied on an obscure paper that a person of skill at the time would be unlikely to have read, since it was published roughly eight years after the '526 patent's priority date. Almeroth Decl., para 80.

245. A person of skill in the art would immediately understand that "recursively traversing" a tree meant using a computer procedure or function that calls itself (on subtrees) with the purpose of visiting every single node in a tree. The '526 patent's teachings are of course nowhere near this. So we must allow the '526 patentees to establish an alternate meaning for this phrase. In my opinion, one of ordinary skill in the art would understand the term "recursively traversing" in the '526 patent to mean searching (in the tree) using a process that repeats itself.

246. Under this understanding, Martinez-Guerra traverses a parse tree. As discussed above, the Martinez-Guerra token recognizer determines, for each input command word, the presence of a matching token in the phrase structure rules of the grammar, which together constitute a command parse tree as described in the '526 patent. *Id.* at 18:31-40. Martinez-Guerra teaches that "[t]oken recognizer 31 performs input matching against each of the valid next states corresponding to a parse state representation consistent with the input stream read so

far." *Id.* at 20:31-34. As successive tokens are matched to potential next legal choices (i.e., "elements") in the representation of the current parse state (*id.* at 20:35-40), the parser-translator updates the current parse state. *Id.* at 20:42-45, 20:49-54. This process of matching input tokens to next legal choices in the phrase structure rules is repeated for each word in the input stream until a phrase structure rule indicates that the input stream has been fully parsed. *Id.* at 20:42-48. Martinez-Guerra thus teaches processing the input stream based on an order of the input command words, namely, the left-to-right order.

247. One of ordinary skill in the art would understand that this process "recursively traverses" the command parse tree in the same way as the '526 patent, namely "traversing using a process that repeats itself." *See* supra at ¶ __. Figure 10 of Martinez-Guerra "illustrates the application of an input token to each parse state" (*id.* at 20:40-41), and this process repeats for each input token in the order received. *Id.* at 10:53-55, 18:62-65. "Additional tokens are parsed from the input stream (e.g., input stream 36) until a complete statement is defined in accordance with the particular grammar employed." *Id.* at 18:62-65. The same process is used in Figure 3 of the '526 patent, where tokens are iteratively matched to elements in the command parse tree. Because Martinez-Guerra uses the same process for matching tokens to next legal choices in the command parse tree as the '526 patent, it "recursively traverses" the command parse tree as claimed by the '526 patent.

Accordingly, it is my opinion that Martinez-Guerra discloses "the determining step includes recursively traversing the command parse tree based on an order of the input command words for identification of the matching token within the identified one element."

c. Non-Asserted Claim 4

248. I understand that claim 4 is not currently asserted against Arista, but claim 6, which depends from claims 2-5, is asserted. Because I understand that the limitations of claim 4 are thus part of claim 6, I will address it below.

> *[4] The method of claim 3, wherein the issuing step includes issuing the prescribed command based on a corresponding command key specified for the matching token within the identified one element.*

249. Martinez-Guerra discloses issuing a prescribed command based on a "translation function identifier" specified for the element that completes a phrase structure rule, such as the "pathname-expert" element of "RULE1." *Id.* at 15:50-62, 16:11-16. The "translation function identifier" specifies where a translation function corresponding to a matched phrase structure rule can be located. *Id.* at 16:11-16, 20:61-64. It is thus a "command key," as claimed in the '526 patent. Martinez-Guerra explains that the translation function identifier "may take the form of . . . an index specifying a function number," and "allows parser-translator component 30 (particularly token recognizer 31 and translator 33 sub-components thereof) to invoke the corresponding . . . translation function in accordance with the grammar 37." *Id.* at 16:16-24. When the input stream contains a legal ordering of tokens according to a phrase structure rule, the translation function corresponding to that phrase structure rule is used to determine the translation. *Id.* at 17:47-52, 20:61-64. The output of the translation process is the "prescribed command."

250. Thus, in the example discussed in claim 1 above, when the final token, "pathname-expert" (which is used in place of the input word "/usr/extract/testing" in the generic command "delete /usr/extract/testing," received from the user) is matched with the "pathname-expert" element of the command parse tree containing "RULE1," a translation function identifier

specified for RULE1 tells the parser-translator where to locate the translation function needed for the translator to issue the prescribed command "rm /usr/extract/testing." For this reason, the prescribed command is based on a corresponding command key specified for the matching token "pathname-expert" within the identified one element.

251. Accordingly, it is my opinion that Martinez-Guerra discloses "the issuing step includes issuing the prescribed command based on a corresponding command key specified for the matching token within the identified one element."

d. Non-Asserted Claim 5

252. I understand that claim 5 is not currently asserted against Arista, but claim 6, which depends from claims 2-5, is asserted. Because I understand that the limitations of claim 5 are thus part of claim 6, I will address it below.

> *[5] The method of claim 4, wherein the issuing step further includes accessing a prescribed translator configured for converting the generic command according to the corresponding command format into the prescribed command based on the corresponding command key.*

253. As discussed above, to the extent Cisco contends that this limitation of the '526 patent reads on the accused products, it is my opinion that Martinez-Guerra discloses claim 5.

254. Martinez-Guerra discloses accessing a prescribed translator (e.g., translator 33) that is configured for converting the generic command according to the corresponding command format (e.g., "delete /usr/extract/testing") into the prescribed command (e.g., "rm /usr/extract/testing") based on the corresponding command key (e.g., translation function identifier corresponding to phrase structure rule RULE1). "A translation function is associated with a phrase structure rule and is invoked by translator 33." *Id.* at 17:51-58. For example, the translation from the generic command "delete /usr/extract/testing" to the prescribed command

"rm /usr/extract/testing" is based on the translation function corresponding to phrase structure rule RULE1. *Id.* at 15:50-62. When a phrase structure rule, such as RULE1, is satisfied by the input words received, Martinez-Guerra explains that the translation corresponding to that rule is performed by a translator component of the parser-translator. *Id.* at 13:29-33, 15:24-28, 17:51-58, 20:61-64. In carrying out such a translation, the prescribed translator for converting the generic command to the prescribed command is necessarily "accessed." As discussed above with reference to claim 4, the accessed translator performs this conversion based on the translation function identifier, which functions as a "command key."

255. Accordingly, it is my opinion that Martinez-Guerra discloses "the issuing step further includes accessing a prescribed translator configured for converting the generic command according to the corresponding command format into the prescribed command based on the corresponding command key."

e. Claim 6

[6] The method of claim 5, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command.

256. Claim 6 is dependent upon non-asserted claim 5, which itself depends from non-asserted claims 4, 3, and 2 respectively. As discussed above, it is my opinion that, under Cisco's apparent interpretation of these claims, Martinez-Guerra discloses all limitations of non-asserted claims 2-5. As discussed above with reference to claim 1, the parser-translator validates the generic command "delete /usr/extract/testing" because the token recognizer determines, for each of the input words "delete" and "/usr/extract/testing" (where "/usr/extract/testing" is matched with the token "pathname-expert" in a dictionary entry), that the token corresponding to each

input word is a legal choice according to the command parse tree representation of the phrase structure rule RULE1. *See id.* at 15:50-62, 18:36-42, 18:66-19:7, 20:31-34, and 10:42-49. By matching the final token ("pathname-expert") to the corresponding element in the command parse tree representation of RULE1 ("pathname-expert"), Martinez-Guerra discloses validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command. Martinez-Guerra satisfies this limitation because validating the entire generic command "delete /usr/extract/testing" necessarily validates "at least a portion of" the generic command. Further, Martinez-Guerra discloses that if an input word is invalid or a command is incomplete—for example, if only the word "delete" is received without any pathname to define the target of the operation—that action may result in returning an error. *See id.* at 19:3-4. In this circumstance, the parser-translator identifies a "best match" for less than the entire command.

257. Martinez-Guerra further issues the prescribed command "rm /usr/extract/testing" based on the identified one element of the command parse tree ("pathname-expert") corresponding to the portion of the generic command ("/usr/extract/testing"). As discussed above with reference to claim 1, "pathname-expert" is the final element in the command parse tree representation of the phrase structure rule RULE1. Martinez-Guerra explains that once the final token in the sequence of legal tokens for a phrase structure rule is matched, the translation function corresponding to the phrase structure rule is used by the translator to output a translation. *Id.* at 9:40-44, 15:24-28, 17:47-52, 20:61-64. Accordingly, when the "pathname-expert" token (which takes the place of the input word "/usr/extract/testing" in the generic command) is matched with the final element "pathname-expert" of RULE1, the input command is fully parsed, and the translation function corresponding to RULE1 is used to issue the

prescribed command "rm /usr/extract/testing." *Id.* at 20:61-64, 21:1-4. Martinez-Guerra explains that this is accomplished by the translator substituting the original input word ("/usr/extract/testing") for the token "pathname-expert" as part of the translation function ("rm <pathname-expert>"). *Id.* at 15:65-16:5. The issued prescribed command is thus "rm /usr/extract/testing." *Id.* at 16:5.

258. Accordingly, it is my opinion that Martinez-Guerra discloses "the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command."

f. Claim 10

[10A] A system configured for executing a plurality of management programs according to respective command formats, the system comprising:

259. In my opinion, Martinez-Guerra discloses the preamble of claim 10. The preamble language of claim 10 is substantially identical to that of claim 1, but is recited as a system instead of a method. As discussed above as to claim 1, Martinez-Guerra discloses a method that is performed by the disclosed system. For this reason, my opinion as to the preamble of claim 1—which I hereby incorporate by reference—applies to claim 10 as well.

[10B] a parser having a command parse tree configured for validating a generic command received from a user, the command parse tree configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the parser identifying one of the elements as a best match relative to the generic command; and

260. As described above with respect to claim 1, the parser-translator of Martinez-Guerra, which includes a parser subcomponent (*id.* at 8:18-23, 9:24-26, Figs. 1-3), is configured to validate generic commands received from the user using phrase structure rules. A person of ordinary skill in the art would understand that for any finite command language, Martinez-Guerra's phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid components of statements in a high-level command language, and because every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed. Martinez-Guerra discloses a user interface that receives from a user an input stream in a high-level language. *Id.* at Fig. 1; 4:4-9; 9:15-23. The parser-translator translates statements in that language into "statements or directives appropriate to a particular data processing application." *Id.* at 4:4-9. One of ordinary skill in the art would understand that the high-level language described in Martinez-Guerra includes "generic commands" within the meaning of the '526 patent, because the input can be an abstraction of a desired operation that is later translated into a directive for a specific software tool. *See id.* at 10:7-11, 13:30-33. For example, Martinez-Guerra discloses that the parser-translator may receive an input stream including the word "delete." *Id.* at 15:50-62 ("the string delete in an input stream such as input stream 36").

261. Martinez-Guerra teaches that the parser builds an internal "parse state data structure" that identifies the "valid next states" according to phrase structure rules encoded in the grammar. *Id.* at 18:31-48. As explained above, Martinez-Guerra's phrase structure rules together constitute a command parse tree because they are collectively a hierarchical data representation of the valid components of statements in a high-level language, and every component of each phrase structure rule has an action that will occur if that component is the last

valid component parsed. The "phrase structure rules" within the grammar specify "the legal orderings of tokens in a language" (*id.* at 14:10-11) and are discussed in the specification of Martinez-Guerra in the context of textual rule specifications, such as "delete pathname-expert." *Id.* at 15:57, 14:32-36.

262. As input words, such as those of the generic command "delete /usr/extract/testing," are received from the user by the parser-translator, they are validated according to the phrase structure rules, which for rules like RULE1 together constitute a command parse tree. As Martinez-Guerra explains, the parser-translator keeps track of next legal choices in the phrase structure rules, and the token recognizer subcomponent of the parser-translator evaluates incoming words received from the user against these next legal choices. *Id.* at 18:66-19:15; 19:25-35, 10:42-49. If the input word matches a next legal choice, then it is validated because it is a permissible choice. *Id.* at 18:66-19:11, 10:42-49.

263. As Martinez-Guerra illustrates, if the input word "delete" is received by the parser-translator, it will match the initial (root level) legal choice for the specification of RULE1 ("delete" in the rule specification "delete pathname-expert") because "delete" is a valid initial command term. *Id.* at 15:50-62. Martinez-Guerra further explains that "parser 32 begins with a first parse state and, consistent with phrase structure rules encoded in grammar 37, collects those tokens that are potentially valid next states corresponding to the first parse state." *Id.* at 19:25-29. Once validated, the token is accepted by the parser, and the parse state is updated to provide the token recognizer with a new list of next legal choices (e.g., tokens that may permissibly follow the token "delete" in the command parse tree representation of RULE1). *Id.* at 19:5-7, 19:16-25, 19:31-35. If the user provides a valid pathname as the second input word, then the parser-translator would validate that token as a next legal choice in the command parse tree

representation of the rule specification of RULE1 (i.e., "pathname-expert" in the rule specification "delete pathname-expert"). *Id.* at 15:59-16:5, 20:40-49. The parser-translator is thus configured to validate a generic command received from the user using a command parse tree.

264. As discussed above with respect to claim 1C, the grammar disclosed by Martinez-Guerra includes "a set of phrase structure rules and associated translation rules that define the syntactic order of a source and target language . . . ." *Id.* at 7:57-59, 16:15-16, 17:51-52. As discussed above, a person of ordinary skill in the art would understand that for any finite command language, Martinez-Guerra's phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid components of statements in a high-level command language, and because every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed. Phrase structure rules like "RULE1" (*id.* at 15:50-62) define a legal ordering of tokens according to a particular language— such as a high-level language containing generic commands (*id.* at 14:11-12)—and can be represented as a textual rule specification ("RULE1 → delete pathname-expert"). In the textual rule specification for RULE1, each token to the right of the arrow in a phrase structure rule is a component needed to form a complete statement; each of these is what the '526 patent calls a "generic command component."

265. Each component of the phrase structure rule has an appropriate corresponding action. If an input is invalid or a command is incomplete—for example, if only the word "delete" is received without any pathname to define the target of the operation, or if the subsequent pathname is wrongly written—that action may result in returning an error. *Id.* at 19:3-4. On the other hand, a valid input stream that matches and completes a phrase structure

rule—e.g., "delete /usr/extract/testing" (id. at 16:5)—will correspond to an action of the translation associated with the phrase structure rule. *Id.* at 8:60-65. The instruction to return an error that results from inputting either an invalid word or an incomplete sequence (e.g., inputting "delete" without a target pathname), or the translation function that results from inputting a complete and valid sequence of tokens (e.g., "delete /usr/extract/testing"), is what the '526 patent refers to as a "command action value." Such a command action value exists for each generic command component of a given phrase structure rule because, as input words are parsed, they may be invalid, resulting in an error, or valid, leading ultimately to the satisfaction of a phrase structure rule, thus resulting in a translation according to the corresponding translation function.

266. Each legal choice according to a phrase structure rule is thus an "element," as described in the '526 patent, having both a generic command component and a corresponding command action value.

267. As discussed above with respect to claim 1C, the token recognizer subcomponent of the parser-translator matches successive input words received from the user with the potential next legal choices in the parse state. *Id.* at 18:38-42, 18:66-19:3, 10:42-49. When the input word matches a next legal choice that completes a phrase—e.g., a valid pathname matching the next legal choice "pathname-expert" in the rule specification RULE1 (*id.* at 15:50-62)—the parser identifies that next legal choice element as a best match for the generic command. *Id.* at 20:45-48.

268. Accordingly, it is my opinion that Martinez-Guerra discloses "a parser having a command parse tree configured for validating a generic command received from a user, the command parse tree configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic

command component and a corresponding at least one command action value, the parser identifying one of the elements as a best match relative to the generic command."

269. Additionally, to the extent Cisco understands the claims to read on Arista product features that do not require that each ingested token have a corresponding translation function, then Martinez-Guerra may anticipates the claim element even if the disclosed error reporting is not a value used to identify a prescribed command.

> *[10C] a plurality of translators configured for issuing commands for the management programs according to respective command formats, the parser outputting a prescribed command to a selected one of the translators based on the identified one element.*

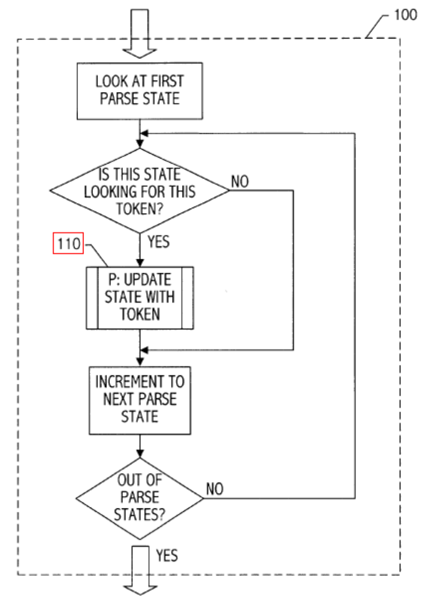
270. In my opinion, Martinez-Guerra discloses this limitation of claim 10.

271. The parser-translator of Martinez-Guerra is configured for issuing commands for a management program according to respective command formats. Martinez-Guerra discloses that multiple implementations of the parser-translator are possible, including one in which "multiple instances of a parser-translator component, each with a corresponding grammar, provide multiple software applications with a common interface to high-level user language statements." *Id.* at 3:43-46. An implementation with multiple parser-translator components includes a "plurality of translators." Likewise, Martinez-Guerra describes an embodiment "wherein operation of [a] single parser-translator component is suitably defined for each software application using a corresponding grammar encoding," *id.* at 3:47-53, which a POSA would understand is functionally equivalent to a plurality of translators.

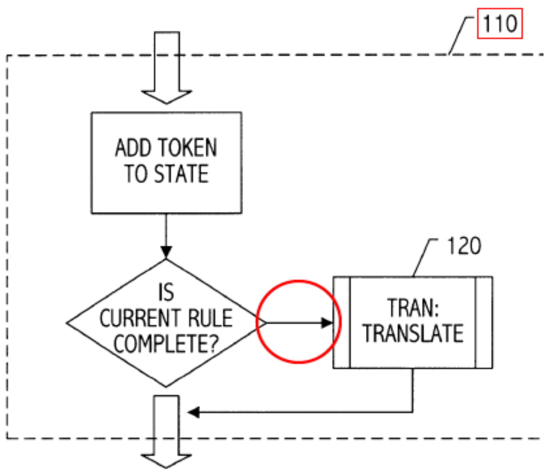
272. As Martinez-Guerra explains, the disclosed translators are configured for issuing commands for management programs according to respective command formats. This is why each of the instances of parser-translators have "a corresponding grammar." *Id.* For example,

Martinez-Guerra discloses that a translator subcomponent of the parser-translator issues an appropriate translation of the generic command "delete /usr/extract/testing" for the UNIX environment as "rm /usr/extract/testing." *Id.* at 15:59-16:5. This is a respective command format of the UNIX environment because "rm" is a UNIX shell command to delete the target file.

273. Martinez-Guerra discloses that once a phrase structure rule is complete, which occurs when the final element of that rule is matched with a token from the input stream (*id.* at 20:45-48), the parser outputs a prescribed command to a translator based on that element. Figures 10 and 11 of Martinez-Guerra illustrate this process. Figure 10 discloses the process for determining whether a token matches the next legal choice of a phrase structure rule. *Id.* at 20:40-42. If the input token is found in the command parse tree containing that phrase structure rule, then the parser updates the parse state by adding the token using process 110 of Figure 10. *Id.* at 20:49-58. Process 110 is illustrated in Figure 11, wherein the parser adds the token to the parse state and determines whether that addition has completed a phrase structure rule (e.g., in the example discussed above, the final token, "pathname-expert," has been received). *Id.* at 20:42-45. If the token completes the rule, Figure 11 illustrates that the parser outputs a prescribed command to a translator, identified by the arrow directed to box 120 ("TRAN:"). *Id.* at 20:40-49. Box 120, illustrated in Figure 12, is the process undertaken by a translator subcomponent of the parser-translator. *Id.* at 20:45-48, Fig. 12.


100
LOOK AT FIRST PARSE STATE
IS THIS STATE LOOKING FOR THIS TOKEN?
NO
YES
110
P: UPDATE STATE WITH TOKEN
INCREMENT TO NEXT PARSE STATE
OUT OF PARSE STATES?
NO
YES

(Fig. 10 (annotation added).)



(Fig. 11 (annotation added).)

274. Because the command represented in Figure 11 is output by the parser as a result of matching the final element of a phrase structure rule, it is output "based on the identified one element." The command is output to the appropriate translator to perform the translation. Martinez-Guerra discloses a "translation function identifier" specified for the element that completes a phrase structure rule, such as the "pathname-expert" element of "RULE1." *Id.* at 15:50-62, 16:11-16. The "translation function identifier" specifies where a translation function corresponding to a matched phrase structure rule can be located. *Id.* at 16:11-16, 20:61-64.

275. Accordingly, it is my opinion that Martinez-Guerra discloses "a plurality of translators configured for issuing commands for the management programs according to respective command formats, the parser outputting a prescribed command to a selected one of the translators based on the identified one element."

g. Claim 11

[11] The system of claim 10, wherein the parser further comprises a command word translation table configured for storing for each

> *prescribed command word a corresponding token for identification of a matching token, the parser configured for determining a presence of the matching token within the command parse tree for each input command word.*

276. In my opinion, Martinez-Guerra discloses claim 11.

277. As described above with respect to claim 2, Martinez-Guerra discloses that the "parser-translator 10 receives an input stream 6 from which token recognizer 11 extracts tokens in accordance with dictionary entries and rules encoded in the grammar 17 and in accordance with a current parse state." *Id.* at 9:35-40. The "dictionary entries" used by the parser-translator "describe all the tokens that a parser-translator . . . should recognize" (*id.* at 14:16-18), and as explained in relation to claim 2 above, parallel the '526 patent's description of the command word translation table "includ[ing] all the command words 26 that are valid according to the generic syntax." '526 patent at 3:45-46. One of ordinary skill in the art would understand from these disclosures and in view of Martinez-Guerra's example of the "delete /usr/extract/testing" generic command, that the dictionary entries are a command word translation table coextensive with the operation of the parser-translator. Martinez-Guerra expressly notes that one of ordinary skill in the art would understand that subcomponent boundaries and their respective functionality in the disclosed parser-translator are subject to compositions other than those described in exemplary embodiments (Martinez-Guerra at 9:26-31, 22:37-42), and as such, would appreciate that the dictionary entries may be part of the parser-translator.

278. As explained with reference to claim 2 above, when the user submits the "delete /usr/extract/testing" command, the first input word "delete" is matched with a "regular token" entry in the dictionary having the same content ("delete"). As Martinez-Guerra explains, a "regular token is a fixed sequence of characters (for example, the string 'delete')." *Id.* at 15:2-4. The next (and final) input word the user submits is the pathname for the file to be deleted

("/usr/extract/testing" in the example). *Id.* at 15:57-16:5. This input word is what Martinez-Guerra refers to as an "expert token," and the parser-translator uses the dictionary entries to match the input word with the token "pathname-expert." *Id.* at 15:4-15, 15:63-16:5. The parser-translator then uses the matching token "pathname-expert" instead of the original input word "/usr/extract/testing" when searching for a match in the command parse tree.

279. As also described above with respect to claim 2, Martinez-Guerra discloses that the parser-translator determines a presence of the matching token within the command parse tree for each input word. In the example discussed above, after the parser-translator has determined the presence of the token corresponding to the input word "delete" in the command parse tree, the parser-translator then seeks to determine the presence of the second (and final) token. Martinez-Guerra explains that this is the token "pathname-expert," corresponding to a valid file path name supplied by the user, as explained above. *Id.* at 15:50-16:5, 20:40-45. The "pathname-expert" token would also be found in the command parse tree because it corresponds to the final token of the phrase structure rule RULE1. *Id.* at 15:57. The parser-translator of Martinez-Guerra thus determines the presence of a matching token in the command parse tree for each input command word.

280. Accordingly, it is my opinion that Martinez-Guerra discloses "the parser further comprises a command word translation table configured for storing for each prescribed command word a corresponding token for identification of a matching token, the parser configured for determining a presence of the matching token within the command parse tree for each input command word."

h. Non-Asserted Claim 12

281. I understand that claim 12 is not currently asserted against Arista, but claim 13, which depends from claim 12, is asserted. Because I understand that the limitations of claim 12 are thus part of claim 13, I will address it below.

> *[12] The system of claim 11, wherein the parser recursively traverses the command parse tree based on an order of the input command words for identification of the matching token within the identified one element.*

282. Non-asserted claim 12 is substantially identical to non-asserted claim 3, which I address above. Accordingly, for the same reasons set forth as to claim 3, it is my opinion that Martinez-Guerra discloses claim 12.

i. Claim 13

> *[13] The system of claim 12, wherein the parser validates at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command.*

283. Claim 13 is dependent upon non-asserted claim 12, which itself depends from claims 11 and 10, respectively. As discussed above, it is my opinion that Martinez-Guerra discloses all limitations of claims 10-12. In my opinion, Martinez-Guerra further discloses the limitations of claim 13.

284. Claim 13 is substantially identical to claim 6, which I address above. Accordingly, for the same reasons set forth as to claim 6, it is my opinion that Martinez-Guerra discloses claim 13.

j. Claim 14

> *[14A] A computer readable medium having stored thereon sequences of instructions for executing a plurality of management programs according to respective command formats, the sequences of instructions including instructions for performing the steps of:*

[14B] receiving a generic command from the user;

[14C] validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format, the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating step including identifying one of the elements as a best match relative to the generic command; and

[14D] issuing a prescribed command of a selected one of the management programs according to the corresponding command format, based on the identified one element.

285. Claim 14 is substantially identical to claim 1, which I address above, except that it is drafted as a "computer readable medium" claim. Martinez-Guerra discloses that its invention may be carried out by instructions embodied on a computer-readable medium. *Id.* at 4:52-56, 5:51-53, 13:55-14:7, 22:45-57, 24:14-34 (claims 16-17), 25:61-26:10 (claims 32-33). Accordingly, for this reason and those set forth as to claim 1, it is my opinion that Martinez-Guerra discloses claim 14.

k. Claim 15

[15A] The medium of claim 14, wherein the generic command includes at least one input command word, the validating step including:

[15B] comparing each input command word to a command word translation table, configured for storing for each prescribed command word a corresponding token, for identification of a matching token; and

[15C] determining a presence of the matching token within the command parse tree for each input command word.

286. Claim 15 is substantially identical to claim 2, which I address above. Accordingly, for the same reasons set forth as to claim 2, it is my opinion that Martinez-Guerra discloses claim 15.

l. Claim 16

[16] The medium of claim 15, wherein the determining step includes recursively traversing the command parse tree based on an order of the input command words for identification of the matching token within the identified one element.

287. Claim 16 is substantially identical to claim 3, which I address above. Accordingly, for the same reasons set forth as to claim 3, it is my opinion that Martinez-Guerra discloses claim 16.

m. Non-Asserted Claim 17

288. I understand that claim 17 is not currently asserted against Arista, but claim 19, which depends from claims 17-18, is asserted. Because I understand that the limitations of claim 17 are thus part of claim 19, I will address it below.

[17] The medium of claim 16, wherein the issuing step includes issuing the prescribed command based on a corresponding command key specified for the matching token within the identified one element.

289. Claim 17 is substantially identical to claim 4, which I address above. Accordingly, for the same reasons set forth as to claim 4, it is my opinion that Martinez-Guerra discloses non-asserted claim 17.

n. Non-Asserted Claim 18

290. I understand that claim 18 is not currently asserted against Arista, but claim 19, which depends from claims 17-18, is asserted. Because I understand that the limitations of claim 18 are thus part of claim 19, I will address it below.

[18] The medium of claim 17, wherein the issuing step further includes accessing a prescribed translator configured for converting the generic command according to the corresponding command format into the prescribed command based on the corresponding command.

291. Claim 18 is substantially identical to claim 5, which I address above. Accordingly, for the same reasons set forth as to claim 5, it is my opinion that Martinez-Guerra discloses non-asserted claim 18.

o. Claim 19

[19]The medium of claim 18, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command.

292. Claim 19 is dependent upon non-asserted claim 18, which itself depends from claims 17, 16, 15, and 14, respectively. As discussed above, it is my opinion that Martinez-Guerra practiced all limitations of claims 14-18. In my opinion, therefore, Martinez-Guerra further practiced the limitations of claim 19.

293. Claim 19 is substantially identical to claim 6, which I address above. Accordingly, for the same reasons set forth as to claim 6, it is my opinion that Martinez-Guerra discloses claim 19.

p. Claim 23

294. Claim 23 is substantially identical to claim 10, which I address above, except that limitation 23B is recited in "means-plus-function" form. Accordingly, in the analysis below, I discuss only claim 23B. For all other limitations, I incorporate by reference my opinion and analysis from the equivalent limitations of claim 10, above.

[23A] A system configured for executing a plurality of management programs according to respective command formats, the system comprising:

> *[23B] means for validating a generic command received from a user, the validating means configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating means identifying one of the elements as a best match relative to the generic command; and*

295. Martinez-Guerra discloses a parser-translator (e.g., parser-translator 10 of Figure 1 and parser-translator 30 of Figure 3) that validates a generic command received from a user. As explained above with reference to claim 10B, Martinez-Guerra discloses that the parser-translator validates tokens—including tokens for a generic command such as "delete /usr/extract/testing"—received from a user via an input stream. *Id.* at Figs. 1, 3; 4:4-9; 9:15-23. The parser-translator of Martinez-Guerra thus includes the claimed "means for validating."

296. As discussed above with respect to claim 1C, Martinez-Guerra discloses that the parser-translator uses a grammar defining "a set of phrase structure rules and associated translation rules that define the syntactic order of a source and target language," (*id.* at 7:57-59, 16:15-16, 17:51-52) and dictionary entries "that describe all the tokens that a parser-translator (e.g., token recognizer 31 of parser-translator component 30) should recognize" (*id.* at 14:16-18). As explained above with reference to claim 10B, the parser-translator is thus configured for specifying valid generic commands relative to a prescribed generic command format.

297. As further discussed above with reference to claim 10B, the command parse tree representation of the phrase structure rules created and maintained by the parser-translator component has elements specifying generic command components (e.g., "delete," "/usr/extract/testing") and a corresponding command action value (e.g., returning an error or performing a translation according to the completed phrase structure rule's corresponding translation function) for each token of a phrase structure rule. *See id.* at 8:60-65, 14:11-12, 14:19-21, 15:50-62, 17:47-58, 18:31-35, 19:3-4.

298. As also explained above with reference to claim 10B, Martinez-Guerra discloses that the parser-translator identifies the element corresponding to the final token needed to complete a phrase structure rule (e.g., "pathname-expert") as the best match relative to the generic command (e.g., "delete /usr/extract/testing"). Because "pathname-expert" is the last token needed to satisfy the phrase structure rule "RULE1," it is the best match relative to the generic command "delete /usr/extract/testing."

299. Accordingly, for these reasons and those explained above as to claim 10, Martinez-Guerra discloses "means for validating a generic command received from a user, the validating means configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating means identifying one of the elements as a best match relative to the generic command."

> *[23C] a plurality of translators configured for issuing commands for the management programs according to respective command formats, the validating means outputting a prescribed command to a selected one of the translators based on the identified one element.*

300. This limitation is identical to that of claim element 10C., and, so, as I previously demonstrated, Martinez-Guerra discloses this limitation.

X. CONCLUSION

301. Based on my knowledge, experience, education, and professional judgment, my understanding of the legal standards in this case, and my review of the evidence, it is my opinion that the Asserted Claims are invalid as anticipated and/or obvious in light of the prior art.

EXHIBIT A

Exhibit A
Douglas W. Clark
Documents Considered List

Claim Construction

- Amended Joint Claim Construction Chart, filed March 5, 2016 (Dkt. 216)
- Declaration of Kevin Almeroth Submitted in Support of Plaintiff Cisco Systems, Opening Claim Construction Brief, filed November 12, 2015

Articles and Books

- Alfred V. Aho, et. al., Compilers: Principles, Techniques, and Tool (1986) [ARISTANDCA00001453]
- Noam Chomsky & George A. Miller, Finite State Languages, 1 Information and Control 91 (1958) (“Chomsky & Miller”)
- Noam Chomsky, On Certain Formal Properties of Grammars, 2 Information and Control 137 (1959)

Patents

- U.S. Patent No. 7,953,886 [ANI-ITC-944_945-1144219]
- U.S. Patent No. 5,832,224 (“Fehskens”) [ARISTANDCA00002286]
- U.S. Patent No. 6,134,709 (“Pratt”),
- U.S. Patent No. 6,523,172 to Martinez-Guerra, et al. (“Martinez-Guerra”) is entitled “Parser Translator System and Method.” [ARISTANDCA00008869]
- U.S. Patent No. 6,724,408 (“Chen”)[ARISTANDCA00000874]
- U.S. Patent No. 7,047,526 (“the ’526 patent”) [CSI-ANI-00129472]

Guides

- Administration for the Avaya Definity Audix System Release 4.0 (May 1999) (“Audix Admin”) [ARISTANDCA00000075]
- Avaya Definity Audix System Release 4.0, Screens Reference (“Definity Audix”) [ARISTANDCA00000447]
- Avaya Definity Audix System Release 4.0, System Description Pocket Reference (May 1999) (“Audix Reference”) [ARISTANDCA00000787]
- Cisco IOS Programmer’s Guide/Architecture Reference, Software Release 12.0 (“ IOS Guide”) [ARISTANDCA00000893]
- JUNOS Internet Software Configuration Guide: Installation and System Management, Release 4.0 (“JUNOS Config Guide”) [ARISTANDCA00002939]
- JUNOS™ Internet Software Configuration Guide: Installation and System Management, Release 4.0 (Mar. 2000) (“JUNOS Software Guide”) [ARISTANDCA000002349]
- Tops-20 User’s Guide [ARISTANDCA00009216]

Exhibit A
Douglas Clark
Documents Considered List

Depositions

- Deposition Transcript of Phil Shafer, taken February 12, 2016
- Deposition Transcript of Prakash Bettadapur, taken May 23, 2016

Miscellaneous

- Decision Before the Patent Trial and Appeal Board Denying Inter Partes Review of Patent 7,047,526
- Lucent Technologies, Getting Started Presentation, Dec. 1999 ("Getting Started")
- And All Materials Cited or Considered in the Attached Expert Report
- Cisco's Amended Infringement Contentions, Exhibit 526-01 (Dec. 14, 2015)

EXHIBIT B

Douglas W. Clark

Department of Computer Science
35 Olden Street
Princeton University
Princeton, NJ 08544-2087

(609) 258-6314
doug@princeton.edu
http://www.cs.princeton.edu/~doug/

2215 St. James Place
Philadelphia, PA 19103
(215) 563-5851

Education

Ph.D. in Computer Science, Carnegie-Mellon University, 1976
Xerox Graduate Fellow, 1973–1976

B.S. *magna cum laude* in Engineering and Applied Science, Yale University, 1972
Yale National Scholar, 1968–1972

Employment

Professor of Computer Science
Princeton University, since 1993

Visiting Professor of Computer and Information Science
University of Pennsylvania, spring 2003

Senior Consulting Engineer
Alpha Advanced Development, Digital Equipment Corp., 1991–1993

Visiting Lecturer
Division of Applied Sciences, Harvard University, 1990–1991

Consulting Engineer, then Senior Consulting Engineer
Advanced VAX Systems Engineering, Digital Equipment Corp., 1982–1990

Principal Engineer
Systems Architecture Group, Digital Equipment Corp., 1980–1982

Member of the Research Staff
Computer Science Laboratory, Xerox Palo Alto Research Center, 1976–1980

Professional activities

International Conference on Computer Design: program committee member, 2003

SIGMETRICS Conference on Measurement and Modeling of Computer Systems: program committee member, 1998, 1991, 1990

NSF Workshop on Critical Issues in Computer Architecture Research: workshop organizer, May 1996

Conference on Architectural Support for Programming Languages and Operating Systems: program chair, 1994; program committee member, 1992, 1988, 1982

ACM Transactions on Computer Systems: associate editor, 1983–1993

NSF Synthesis Engineering Education Coalition: national advisory committee member, 1990–1993

NSF Division of Microelectronic Information Processing Systems: advisory committee member, 1989–1992

International Symposium on Computer Architecture: program committee member, 1990, 1985; program committee co-chair, 1986

Publications

Wu, Q. Reddi, V., Wu, Y. Lee, J., Connors, D., Brooks, D., Martonosi, M., and Clark, D.W. "Dynamic Compiler Driven Control for Microprocessor Energy and Performance," *IEEE Micro* Special Issue: Top Picks from Computer Architecture Conferences, Vol. 26, No. 1, Jan./Feb., 2006.

Wu, Q. Reddi, V., Wu, Y. Lee, J., Connors, D., Brooks, D., Martonosi, M., and Clark, D.W. "A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance," *Proc. 38th IEEE/ACM International Symposium on Microarchitecture* (MICRO-38), Barcelona, Spain, November 2005. Best Paper Award.

Wu, Q., Juang, P., Peh, L.-S., Martonosi, M. and Clark, D.W. "Formal Control Techniques for Power-Performance Management," *IEEE Micro*, Vol 25, No. 5, Sept./Oct. 2005.

Juang, P., Wu, Q., Peh, L.-S., Martonosi, M. and Clark, D.W. "Coordinated, Distributed, Formal Energy Management of Chip Multiprocessors," *Proc. International Symposium on Low Power Electronics and Design* (ISLPED-05), San Diego, Aug. 2005.

Wallace, G., Anshus, O.J., Bi, P., Chen, H., Chen, Y., Clark, D., Cook, P., Finkelstein, A., Funkhouser, T., Gupta, A., Hibbs, M., Li, K., Liu, Z., Samanta, R., Sukthankar, R., and Troyanskaya, O. "Tools and Applications for Large-Scale Display Walls," *IEEE Computer Graphics*, Vol. 25, No. 4, July/Aug. 2005.

Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Voltage and Frequency Control with Adaptive Reaction Time in Multiple-Clock-Domain Processors," *Proc. 11th International Symposium on High-Performance Computer Architecture* (HPCA-11), San Francisco, Feb. 2005.

Wu, Q., Juang, P., Martonosi, M., and Clark, D.W. "Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors," *Proc. Eleventh Int. Conf. on Architectural Support for*

Programming Languages and Operating Systems (ASPLOS-XI), Boston, Oct. 2004.

Y. Zhou, Y., Wang, L., Clark, D.W., and K. Li. "Thread Scheduling for Out-of-Core Applications with a Memory Server," ch. 9, *Scalable Input/Output: Achieving System Balance*, Daniel A. Reed, ed. MIT Press, 2004, pp. 233–253.

Juang, P., Skadron, K., Martonosi, M., Hu, Z., Clark, D.W, Diodato, P., and Kaxiras, S. "Implementing Branch-Predictor Decay Using Quasi-Static Memory Cells," *ACM Transactions on Architecture and Code Optimization*, Vol. 1, No. 2, June 2004.

Wu, Q., Pyatakov, A., Spiridonov, A., Raman, E., Clark, D.W., and August, D. "Exposing Memory Access Regularities Using Object-Relative Memory Profiling," *Proc. Second Ann. IEEE/ACM Int. Symp. on Code Generation and Optimization (CGO 2004)*, San Jose, March 2004.

Juang, P., Diodato, P., Kaxiras, S., Skadron, K., Hu, Z., Martonosi, M., and Clark, D.W. "Implementing Decay Techniques using 4T Quasi-Static Memory Cells," *Computer Architecture Letters 1*, Sept. 2002.

Hu, Z., Juang, P., Skadron, K., Martonosi, M., and Clark, D.W. "Applying Decay Strategies to Branch Predictors for Leakage Energy Savings," *Proc. 2002 Int. Conf. on Computer Design (ICCD2002)*, Sept. 2002.

Hu, Z., Juang, P., Diodato, P., Kaxiras, S., Skadron, K., Martonosi, M., and Clark, D.W. "Managing Leakage for Transient Data: Decay and Quasi-Static 4T Memory Cells," *Proc. 2002 Int. Symp. on Low Power Electronics and Design (ISLPED 2002)*, Aug. 2002.

Chen, Y., Chen, H., Clark, D.W., Liu, Z., Wallace, G., and Li, K. "Software Environments for Cluster-Based Display Systems," *IEEE Int. Symp. on Cluster Computing and the Grid (CCGrid 2001)*, May 2001.

Chen, Y., Clark, D.W., Finkelstein, A., Housel, T., and Li, K. "Automatic Alignment Of High-Resolution Multi-Projector Displays Using An Un-Calibrated Camera," *IEEE Visualization 2000*, Salt Lake City, Oct. 2000.

Skadron, K., Martonosi, M., and Clark, D.W. "A Taxonomy of Branch Mispredictions, and Alloyed Prediction as a Robust Solution to Wrong-History Mispredictions," *Int. Conf. on Parallel Architectures and Compilation Techniques*, Philadelphia, Oct. 2000.

Li, K., Chen, H., Chen, Y., Clark, D.W., Cook, P., Damianakis, S., Essl, G., Finkelstein, A., Funkhouser, T., Housel, T., Klein, A., Liu, Z., Praun, E., Samanta, R., Shedd, B., Singh, J.P., Tzanetakis, G., and Zheng, J. "Building and Using a Scalable Display Wall System," *IEEE Computer Graphics and Applications 20*, 4 (July/Aug. 2000), pp. 29-37.

Skadron, K., Martonosi, M., and Clark, D.W. "Speculative Updates of Local and Global Branch History: A Quantitative Analysis," *The Journal of Instruction Level Parallelism*, vol. 2, January 2000 (http://www.jilp.org/vol2).

Skadron, K., Ahuja, P.S., Martonosi, M., and Clark, D.W. "Branch Prediction, Instruction-Window Size, and Cache Size: Performance Tradeoffs and Sampling Techniques," *IEEE Transactions on Computers 48*, 3 (Nov. 1999), pp. 1260-1281.

Martonosi, M., Karlin, S., Liao, C., and Clark, D.W. "Performance Monitoring Infrastructure in the Shrimp Multicomputers," *Int. Journal of Parallel and Distributed Systems and Networks 2*, 3 (1999), pp. 126-133.

Liao, C., Martonosi, M., and Clark, D.W. "Experience with an Adaptive Globally-Synchronizing Clock Algorithm," *Proc. 11th ACM Symp. on Parallel Algorithms and Architectures*, Saint-Malo, France, June 1999.

Liao, C., Martonosi, M., and Clark, D.W. "An Adaptive Globally-Synchronizing Clock Algorithm and its Implementation on a Myrinet-based PC Cluster," *Proc. SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, Atlanta, May 1999.

Zhou, Y., Wang, L., Clark, D.W., and Li, K. "Thread Scheduling for Out-of-Core Applications with Memory Server on Multicomputers," *Proc. 6th Workshop on I/O in Parallel and Distributed Systems*, Atlanta, May 1999.

Skadron, K., Ahuja, P.S., Martonosi, M., and Clark, D.W. "Improving Prediction for Procedure Returns with Return-Address-Stack Repair Mechanisms," *Proc. 31st Annual Int. Symposium on Microarchitecture*, Dallas, Dec. 1998.

Wei, B., Clark, D.W., Felten, E.W., Li, K., and Stoll, G. "Performance Issues of a Distributed Frame Buffer on a Multicomputer," *Proc. 1998 Eurographics/SIGGRAPH Workshop on Graphics Hardware*, Lisbon, Aug./Sept. 1998.

Liao, C., Martonosi, M., and Clark, D.W. "Performance Monitoring in a Myrinet-Connected Shrimp Cluster," *Proc. 2nd SIGMETRICS Symposium on Parallel and Distributed Tools*, Oregon, August 1998.

Ahuja, P.S., Skadron, K., Martonosi, M., and Clark, D.W. "Multipath Execution: Opportunities and Limits," *Proc. 12th International Conference on Supercomputing*, Melbourne, July 1998.

Liao, C., Jiang, D., Iftode, L., Martonosi, M., and Clark, D.W. "Monitoring Shared Virtual Memory Performance on a Myrinet-based PC Cluster," *Proc. 12th International Conference on Supercomputing*, Melbourne, July 1998.

Blumrich, M.A., Alpert, R.D., Chen, Y., Clark, D.W., Damianakis, S.N., Dubnicki, C., Felten, E.W., Iftode, L., Li, K., Martonosi, M., and Shillner, R.A. "Design Choices in the SHRIMP System: An Empirical Study," *Proc. 25th Intl. Symposium on Computer Architecture*, Barcelona, June 1998, pp. 330-341.

Leung, C., Brooks, D., Martonosi, M., and Clark, D.W. "Power-Aware Architecture Studies: Ongoing Work at Princeton," *Proc. Power-Driven Microprocessor Design Workshop*, Barcelona, June 1998.

Wei, B., Stoll, G., Felten, E.F., Clark, D.W., and Li, K. "RAIN: Supporting Parallel Graphics on Paragon Multicomputers," *Proc. 13th Ann. Conf., Intel Supercomputer Users Group*, Albuquerque, June 1997.

Skadron, K. and Clark, D.W. "Design Issues and Trade-offs for Write Buffers," *Proc. Third Intl. Symposium on High-Performance Computer Architecture*, San Antonio, Feb. 1997, pp. 144-155.

Martonosi, M., Clark, D.W., and Mesarina, M., "The SHRIMP Hardware Performance Monitor: Design and Applications," *Proc. SIGMETRICS Symposium on Parallel and Distributed Tools*, Philadelphia, May 1996.

Felten, E.W., Alpert, R.D., Bilas, A., Blumrich, M.A., Clark, D.W., Damianakis, S., Dubnicki, C., Iftode, L., and Li, K., "Early Experience with Message-Passing on the SHRIMP Multicomputer," *Proc. 23rd Intl. Symposium on Computer Architecture*, Philadelphia, May 1996, pp. 296-307.

Skadron, K. and Clark, D.W., "Measuring the Effects of Retirement and Load-Service Policies on Write Buffer Performance," *Proc. 1996 Workshop on Performance Analysis and its Impact on Design (PAID)*, IBM Austin Research Lab, Austin, March 1996.

Ahuja, P.S., Clark, D.W., and Rogers, A., "The Performance Impact of Incomplete Bypassing in Processor Pipelines," *28th IEEE/ACM Annual Int. Symp. on Micro-Arch. (MICRO-28)*, Ann Arbor, Nov. 1995, pp. 36-45.

Wei, B., Stoll, G., Clark, D.W., Felten, E.W., Li, K., and Hanrahan, P., "Synchronization for a Multi-Port Frame Buffer on a Mesh-Connected Multicomputer," *Proc. Parallel Rendering Symposium*, Atlanta, Oct. 1995, pp. 81-88.

Stoll, G., Wei, B., Clark, D.W., Felten, E.W., Li, K., and Hanrahan, P., "Evaluating Multi-Port Frame Buffer Designs for a Mesh-Connected Multicomputer," *Proc. 22nd International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, June 1995, pp 96-105.

Clark, D.W. and Weng, L.-J., "Maximal and Near-Maximal Shift Register Sequences: Efficient Event Counters and Easy Discrete Logarithms," *IEEE Transactions on Computers 43*, 5 (May 1994), pp. 560-568.

Bhandarkar, D. and Clark, D.W., "Performance from Architecture: Comparing a RISC and a CISC with Similar Hardware Organization," *Proc. Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, Santa Clara, CA, April 1991, pp. 310-319.

Clark, D.W., "Large-Scale Hardware Simulation: Modeling and Verification Strategies" (invited paper), Chapter 9 of *CMU Computer Science: A 25th Anniversary Commemorative* (R.F. Rashid, ed.), ACM Press/Addison-Wesley, 1991, pp. 219-234.

Clark, D.W., "Bugs are Good: A Problem-Oriented Approach to the Management of Design Engineering," *Research-Technology Management 33*, 3 (May-June 1990), pp. 23-27.

Clark, D.W., Bannon, P.J., and Keller, J.B., "Measuring VAX 8800 Performance with a Histogram Hardware Monitor," *Proc. 15th International Symposium on Computer Architecture*, Honolulu, June 1988, pp. 176-185.

Clark, D.W., "Pipelining and Performance in the VAX 8800," *Proc. Second International Symposium on Architectural Support for Programming Languages and Operating Systems*, Palo Alto, Oct. 1987, pp. 173-177.

Clark, D.W. and Emer, J.S., "Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement," *ACM Transactions on Computer Systems 3*, 1 (Feb. 1985), pp. 31-62.

Emer, J.S. and Clark, D.W., "A Characterization of Processor Performance in the VAX-11/780," *Proc. 11th International Symposium on Computer Architecture*, Ann Arbor, June 1984, pp. 301-310.

Clark, D.W., "Cache Performance in the VAX-11/780," *ACM Transactions on Computer Systems 1*, 1 (Feb. 1983), pp. 24-37.

Levy, H.M. and Clark, D.W., "On the Use of Benchmarks for Measuring System Performance," *ACM Computer Architecture News 10*, 6 (Dec. 1982), pp. 5-8.

Clark, D.W. and Levy, H.M., "Measurement and Analysis of Instruction Use in the VAX-11/780," *Proc. 9th International Symposium on Computer Architecture*, Austin, April 1982, pp. 9-17.

Clark, D.W., Lampson, B.W., and Pier, K.A., "The Memory System of a High-Performance Personal Computer," *IEEE Transactions on Computers C-30*, 10 (Oct. 1981), pp. 715-733.

Clark, D.W. and Strecker, W.D., “Comments on ‘The Case for the Reduced Instruction Set Computer’ by Patterson and Ditzel,” *ACM Computer Architecture News 8*, 6 (Oct. 1980), pp. 34-38.

Bobrow, D.G. and Clark, D.W., “Compact Encodings of List Structure,” *ACM Transactions on Programming Languages and Systems 1*, 2 (Oct. 1979), pp. 266-286.

Clark, D.W., “Measurements of Dynamic List Structure Use in Lisp,” *IEEE Transactions on Software Engineering SE-5*, 1 (Jan. 1979), pp. 51-59.

Clark, D.W. and Green, C.C., “A Note on Shared List Structure in Lisp,” *Information Processing Letters 7*, 6 (Oct. 1978), pp. 312-314.

Clark, D.W., “A Fast Algorithm for Copying List Structures,” *Communications of the ACM 21*, 5 (May 1978), pp. 351-357.

Clark, D.W. and Green, C.C., “An Empirical Study of List Structure in Lisp,” *Communications of the ACM 20*, 2 (Feb. 1977), pp. 78-87.

Clark, D.W., *List Structure: Measurements, Algorithms, and Encodings*, Ph.D. thesis, Dept. of Computer Science, Carnegie-Mellon University, Aug. 1976.

Clark, D.W., “An Efficient List-Moving Algorithm Using Constant Workspace,” *Communications of the ACM 19*, 6 (June 1976), pp. 352-354.

Clark, D.W., “A Fast Algorithm for Copying Binary Trees,” *Information Processing Letters 4*, 3 (Dec. 1975), pp. 62-63.

Clark, D.W., “Hardware Systems in the Core Curriculum of a Computer Science Ph.D. Program,” *Proc. ACM Fourth Technical Symposium on Computer Science Education*, Detroit, Feb. 1974, pp. 106-110.

EXHIBIT C

Amended Joint Claim Construction Chart

The parties prepared this amended chart in response to the Court's request to list no more than ten claim terms.

Cisco's Statement

Cisco notes that Arista has made changes to certain of its proposed constructions and evidence, which were not addressed as part of the claim construction briefing. Cisco has not had the opportunity to consider these changes. Cisco reserves all rights to object to these changes and/or to supplement its positions or evidence in light of these changes by Arista. For example, to the extent that Arista is permitted to make arguments based on Cisco's IPR filings, Cisco reserves the right to make arguments based on Arista's IPR filings relating to the patents-in-suit.

Arista's Statement

Arista's new citations are limited to intrinsic evidence that did not exist during the claim construction briefing. Specifically, Arista added citations to four pages of Cisco's Preliminary Patent Owner Response, which Cisco filed in connection with the IPR on the '526 patent on February 18, 2016.

I. ’526 Patent

Term	**Cisco’s Proposed Construction And Evidence**	**Arista’s Proposed Construction And Evidence**
A. “management programs”	*Proposed construction:* “separate tools or external agents having their own respective command formats that provide management functions” *Intrinsic Evidence:* The specification of the ’526 patent, including: Abstract, col. 1:54-63; 1:64-68; 2:8-12; 2:13-15; 2:24-29; 3:1-15; Figures 1, 3. The prosecution history of the ’526 patent, including prior art references cited therein, including: File History, Rejection, 1/15/2003; File History, Applicant Response, 4/16/2003; File History, Applicant Response, 10/7/2003. *Extrinsic Evidence:* None	*Proposed construction:* “tools that are configured to execute user-entered commands having their own respective command formats rather than the generic command format” This proposal assumes that the term “generic command format” is not found to be indefinite. *Intrinsic Evidence:* Abstract; Figs. 1-3; col. 1:6-63, 2:57-3:35; 4:19-36; 4:55-60; Appx. A & B. File History, 10/3/2003 Response to Office Action; 2/18/2004 Response to Office Action; 3/18/2004 Office Action; 6/18/2004 Response to Office Action; 11/11/2004 Office Action. *Extrinsic Evidence:* None

Term	**Cisco's Proposed Construction And Evidence**	**Arista's Proposed Construction And Evidence**
B. "generic command"	*Proposed construction:* "command that provides an abstraction of the tool-specific command formats and syntax, enabling a user to issue the command based on the relative functions, as opposed to the specific syntax for a corresponding tool" *Intrinsic Evidence:* The specification of the '526 patent, including: Title, Abstract; col. 1:58-63, 3:27-35, 3:47-51, 3:54-61; Figures 1, 2, 3. The prosecution history of the '526 patent, including prior art references cited therein, including: File History, Response after Final Rejection, 2/18/2004; File History, Rejection, 3/18/2004; File History, Applicant Response, 6/23/2004. *Extrinsic Evidence:* <u>Webster's Third New International Dictionary, 2002</u>: entry for "generic."	*Proposed construction:* Indefinite, OR if not indefinite: "command having a format and syntax that is an abstraction of the command formats and syntaxes of more than one management program, as opposed to the specific syntax for any such management program" *Intrinsic Evidence:* Abstract; Figs. 1-3; col. 1:6-63; 2:57-3:35; 4:19-36; 4:55-60; Appx. A & B. File History, 10/3/2003 Response to Office Action; 2/18/2004 Response to Office Action; 3/18/2004 Office Action; 6/18/2004 Response to Office Action; 11/11/2004 Office Action. *Extrinsic Evidence:* None

Term	Cisco's Proposed Construction And Evidence	Arista's Proposed Construction And Evidence
C. "command parse tree"	*Proposed construction:* "a hierarchical data representation having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value" *Intrinsic Evidence:* The specification of the '526 patent, including: Abstract; col. 1:48-54; 1:67-2:8; 2:15-24; 3:36-39; 3:47-57; 4:10-18; Figures 2, 3. The prosecution history of the '526 patent, including prior art references cited therein, including: File History, Rejection, 3/18/2004; File History, Applicant Response, 6/23/2004. *Extrinsic Evidence:* Alice E. Fischer et al., The Anatomy of Programming Languages 74-75 (1993). Allen I. Holub, Compiler Design In C 4 (1999). Microsoft Computer Dictionary 529 (5th ed. 2002). Computer Desktop Encyclopedia 991 (9th ed. 2001).	*Proposed construction:* "tree": "data structure consisting of linked nodes, with a root node (a node with no parent nodes), and where the remaining nodes are either a branch node (a node with a parent node and one or more children nodes), or a leaf node (a node with a parent node and no children nodes)" "command parse tree": "tree for interpreting commands where each node, or element, corresponds to one or more command components"[1] *Intrinsic Evidence:* Abstract; Figs. 2-3; col. 3:36-4:54, Appx. B. File History, 10/3/2003 Response to Office Action; 2/18/2004 Response to Office Action; 3/18/2004 Office Action; 6/18/2004 Response to Office Action; 11/11/2004 Office Action. *Extrinsic Evidence:* Microsoft Press Computer Dictionary, 3d ed. (1997): entry for "tree." Barron's Dictionary of Computer and Internet Terms, 6th ed. (1998): entry for "tree." Random House Webster's Computer & Internet Dictionary (1999): entry for "tree structure."

[1] Arista initially proposed the following construction for "command parse tree": "tree for interpreting commands where each node corresponds to a command component." In response to Cisco's argument that a node or element may correspond to "at least one" (as opposed to just one) command component (Doc. 91 at 9), Arista has modified its construction as set forth above.

Term	**Cisco's Proposed Construction And Evidence**	**Arista's Proposed Construction And Evidence**
D. "the validating step including identifying one of the elements as a best match relative to the generic command"	*Proposed construction:* Plain and ordinary meaning (except that specific terms appearing within the phrase should be construed as proposed above) *Intrinsic Evidence:* The specification of the '526 patent, including: col. 1:64-2:8; Figure 2; Claims 1, 14. The prosecution history of the '526 patent, including prior art references cited therein. *Extrinsic Evidence:* None	*Proposed construction:* Indefinite, OR if not indefinite: "the validating step having the capability of both identifying the element in the parse tree that exactly matches the generic command, and, in the absence of an exact match, identifying the element that contains the last validated component of the generic command" *Intrinsic Evidence:* Abstract; Figs. 2-3; 3:36-4:54; Appx. A & B Patent Owner Preliminary Response, *Arista Networks, Inc., Petitioner v. Cisco Systems, Inc., Patent Owner*, Case IPR2016-00119, at 8–9, 27–28. *Extrinsic Evidence:* None

Term	**Cisco's Proposed Construction And Evidence**	**Arista's Proposed Construction And Evidence**
E. "the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value"	*Proposed construction:* Plain and ordinary meaning (except that specific terms appearing within the phrase should be construed as proposed above) *Intrinsic Evidence:* The specification of the '526 patent, including: col. 1:64-2:8; Figures 1, 2, 3; Claims 1, 14. The prosecution history of the '526 patent, including prior art references cited therein. *Extrinsic Evidence:* None	*Proposed construction:* "elements": "nodes" "command action value": "piece of data that uniquely represents the prescribed command." the entire phrase: "the command parse tree having nodes, such that each node specifies a unique command action value for each generic command component." *Intrinsic Evidence:* Abstract; Figs. 2-3; col. 3:36-4:54; Appx. A & B. File History, 10/3/2003 Response to Office Action; 12/18/2003 Office Action; 6/18/2004 Response to Office Action; 11/11/2004 Office Action. Patent Owner Preliminary Response, *Arista Networks, Inc., Petitioner v. Cisco Systems, Inc., Patent Owner*, Case IPR2016-00119, at 8–9, 27–28. *Extrinsic Evidence:* None

Term	**Cisco's Proposed Construction And Evidence**	**Arista's Proposed Construction And Evidence**
F. "means for validating a generic command received from a user, the validating means configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating means identifying one of the elements as a best match relative to the generic command"	*Proposed construction:* Function: validating a generic command received from a user Structure: Parser 14 in Figure 2, which includes the command word translation table 20 and the command parse tree 22, as described in 3:36-61, and equivalents *Intrinsic Evidence:* The specification of the '526 patent, including: col. 3:36-61; Figure 2. The prosecution history of the '526 patent, including prior art references cited therein, including: File History, Applicant Response, 6/23/2004. *Extrinsic Evidence:* Webster's Third New International Dictionary, 2002: entry for "validate."	*Proposed construction:* Functions: (1) validating a generic command received from a user (2) specifying valid generic commands relative to a prescribed generic command format, (3) having elements each specifying at least one corresponding generic component and a corresponding at least one command action value, and (4) identifying one of the elements as a best match relative to the generic command. Disclosed structure: A processor executing a parser, and a corresponding memory storing a command parse tree, wherein the parser executes the algorithm of Figure 3, and wherein (1) each node of the command parse tree specifies one token and a corresponding command key; (2) the top-level nodes of the command parse tree represent all possible valid first words in the input command, second-level nodes represent all possible valid second words for each valid first word in the input command, and so on; *Intrinsic Evidence:* Abstract; Figs. 2-3; col. 3:36-4:54; Appx. A & B. *Extrinsic Evidence:* None

II. ’886 Patent

Term	Cisco’s Proposed Construction And Evidence	Arista’s Proposed Construction And Evidence
A. “extensible markup language (XML)”	*Proposed construction:* “extensible”: a property of a computer language that allows the user to add new features or modify existing ones “markup language”: a computer language that allows the user to add identifiers to a document for indicating logical components or layout *Intrinsic Evidence:* The specification of the ’886 patent, including col. 3:10-7:21. The prosecution history of the ’886 patent, including prior art references cited therein. *Extrinsic Evidence:* McGraw-Hill Dictionary of Scientific and Technical Terms, 6th Ed (2003): entries for “Extensible language,” “markup,” and “markup language.” Microsoft Computer Dictionary, 5th Ed. (2002): entry for “markup language.” Dictionary of computer science, engineering, and technology (2001): entry for “markup language.”	*Proposed construction:* “markup language defined by one of the versions of the XML standard published by the W3C organization” *Intrinsic Evidence:* Figs. 3 & 4; col. 1:49-57; 3:11-36; 4:24–42; 4:50-5:65; 6:12-7:17. File History, 9/4/2008 Response to Office Action; 10/17/2008 Office Action; 1/21/2009 Response to Office Action; 3/13/2009 Office Action; 6/15/2009 Response to Office Action; 12/8/2010 Response to Office Action. *Extrinsic Evidence:* Computer Desktop Encyclopedia, 9th Ed. (2001): entry for "XML (Extensible Markup Language)". Webster’s New World Computer Dictionary, 9th Ed. (2001), 10th Ed. (2003): entry for "“XML Abbreviation for Extensible Markup Language." Microsoft Computer Dictionary, Fifth Edition (2002): entry for "XML." Barron’s Dictionary of Computer and Internet Terms, 8th Ed. (2003): entry for "XML (Extensible Markup Language)." See generally http://www.w3.org/standards/xml/.

Term	**Cisco's Proposed Construction And Evidence**	**Arista's Proposed Construction And Evidence**
B. Cisco's proposed term: "command line interface (CLI) parser" Arista's proposed term: "receiving, with a command line interface (CLI) parser"	*Proposed construction:* "command line interface (CLI) parser": "a component of the routing system for analyzing command line interface (CLI) commands using a grammar" *Intrinsic Evidence:* The specification of the '886 patent, including Abstract, col. 1:41-64; 2:52-5:9; 6:11-24. The prosecution history of the '886 patent, including prior art references cited therein, including Amendments filed on 1/19/2010, 12/8/2010, and cited reference U.S. Patent No. 5,778,223. *Extrinsic Evidence:* Dictionary of computer science, engineering, and technology (2001): entry for "parsing." McGraw-Hill Dictionary of Scientific and Technical Terms, 6th Ed (2003): entry for "parsing." Wiley electrical and electronics engineering dictionary (2004): entry for "parse."	*Proposed construction:* "receiving": "taking as an input" "a command line interface (CLI) parser": "a program that breaks down the individual sub-parts of a command using a CLI grammar" *Intrinsic Evidence:* Abstract, Figs. 1-4; Cols. 1:40-57; 1:61-2:5; 2:52-3:45; 3:60-4:33; 4:60-65. File History, 9/4/2008 Response to Office Action; 10/17/2008 Office Action; 1/21/2009 Response to Office Action; 3/13/2009 Office Action; 6/15/2009 Response to Office Action; 1/19/2010 Response to Office Action; 12/8/2010 Response to Office Action. *Extrinsic Evidence:* McGraw-Hill Dictionary of Computing and Communications (2003): entries for "parser" and "parsing." Dictionary of Computer and Internet Words, Houghton Mifflin Co. (2001): entry for "parse." Webster's New World Computer Dictionary, 9th Ed. (2001), 10th Ed. (2003): entry for "parser." Barron's Dictionary of Computer and Internet Terms, 8th Ed. (2003): entry for "parsing." Oxford Dictionary of Computing, 5th Ed. (2004): entry for "parsing."

Case 5:14-cv-05344-BLF Document 216 Filed 03/05/16 Page 10 of 13

Term	**Cisco's Proposed Construction And Evidence**	**Arista's Proposed Construction And Evidence**
		Dictionary of Computing, 5th Ed. (2004): entries for "parse" and "parser." Microsoft Computer Dictionary, Fifth Edition (2002): entries for "parse" and "parser." Computer Desktop Encyclopedia, 9th Ed. (2001): entries for "parse" and "parser."

Term	**Cisco's Proposed Construction And Evidence**	**Arista's Proposed Construction And Evidence**
C. "parsing the output message to identify at least one CLI token"	*Proposed construction:* analyzing the output message to extract at least one unit of CLI characters in a sequence *Intrinsic Evidence:* The specification of the '886 patent, including col. 6:30- 7:21. The prosecution history of the '886 patent, including prior art references cited therein, including Amendments filed on 1/21/2009 and 6/15/2009. *Extrinsic Evidence:* McGraw-Hill Dictionary of Scientific and Technical Terms, 6th Ed (2003): entry for "token." Dictionary of computer science, engineering, and technology (2001): entry for "parsing."	*Proposed construction:* "breaking down the output message into its constituent character strings and determining that at least one such string corresponds to a CLI keyword or parameter" *Intrinsic Evidence:* Fig. 4; Tbl. 3; col 1:54-57; 6:24-49. File History: 6/15/2009 Response to Office Action. *Extrinsic Evidence:* None

Term	**Cisco's Proposed Construction And Evidence**	**Arista's Proposed Construction And Evidence**
D. "wherein the input command is configured in an extensible markup language (XML) format having a CLI syntax with CLI keywords sequenced according to configuration rules for CLI commands"	*Proposed construction:* Plain and ordinary meaning, except for "extensible markup language (XML) format" as construed below. *Intrinsic Evidence:* The specification of the '886 patent, including at col. 3:10-5:65. The prosecution history of the '886 patent, including prior art references cited therein. *Extrinsic Evidence:* <u>McGraw-Hill Dictionary of Scientific and Technical Terms, 6th Ed (2003)</u>: entries for "Extensible language," "markup," and "markup language." <u>Microsoft Computer Dictionary, 5th Ed. (2002)</u>: entry for "markup language." <u>Dictionary of computer science, engineering, and technology (2001)</u>: entry for "markup language."	*Proposed construction:* "an extensible markup language (XML) format": "a format that complies with one of the versions of the XML standard published by the W3C organization" "keyword": "word describing an action or operation that the computer can recognize and execute" The entire phrase: "wherein the input command is written in an extensible markup language (XML) format, such that one or more CLI keywords, on the one hand, and the CLI parameters, on the other, are contained within respective XML tags, and the sequence of the tags complies with the sequencing rules for keywords and parameters in the CLI grammar and syntax" *Intrinsic Evidence:* Figs. 3 & 4; cols. 1:49-57; 3:11-36; 4:24–42; 4:50-5:65; 6:12-7:17. File History: 9/4/2008 Response to Office Action; 10/17/2008 Office Action; 1/21/2009 Response to Office Action; 3/13/2009 Office Action; 6/15/2009 Response to Office Action; 12/8/2010 Response to Office Action. *Extrinsic Evidence:* <u>Computer Desktop Encyclopedia, 9th Ed. (2001)</u>: entry for "XML (Extensible Markup Language)." <u>Webster's New World Computer Dictionary, 9th Ed. (2001), 10th Ed. (2003)</u>: entry for ""XML Abbreviation for Extensible Markup Language." <u>Microsoft Computer Dictionary, Fifth Edition (2002)</u>: entry for

Case 5:14-cv-05344-BLF Document 216 Filed 03/05/16 Page 13 of 13

Term	Cisco's Proposed Construction And Evidence	Arista's Proposed Construction And Evidence
		"XML." Barron's Dictionary of Computer and Internet Terms, 8th Ed. (2003): entry for "XML (Extensible Markup Language)." See generally http://www.w3.org/standards/xml/. Webster's New World Computer Dictionary, 9th Ed. (2001), 10th Ed. (2003): entry for "keyword." Dictionary of Computer and Internet Words, Houghton Mifflin Co. (2001): entry for "keyword." Computer Desktop Encyclopedia, 9th Ed. (2001): entry for "keyword." Dictionary of Computing, 5th Ed. (2004): entry for "keyword."

EXHIBIT D

US007047526B1

(12) **United States Patent**
Wheeler et al.

(10) **Patent No.: US 7,047,526 B1**
(45) **Date of Patent: *May 16, 2006**

(54) **GENERIC COMMAND INTERFACE FOR MULTIPLE EXECUTABLE ROUTINES**

(75) Inventors: **Jeffrey Wheeler**, Glen Allen, VA (US); **Paul Mustoe**, Midlothian, VA (US)

(73) Assignee: **Cisco Technology, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 918 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/604,880**

(22) Filed: **Jun. 28, 2000**

(51) **Int. Cl.**
G06F 9/44 (2006.01)

(52) **U.S. Cl.** **717/143**; 717/144; 717/136; 717/127; 707/3; 707/10; 707/100; 719/320

(58) **Field of Classification Search** 717/136–167, 717/127; 707/100, 3, 10; 719/320
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,837,798 A		6/1989	Cohen et al.	
5,086,504 A	*	2/1992	Nemeth-Johannes et al.	717/143
5,379,419 A	*	1/1995	Heffernan et al	707/4
5,491,796 A	*	2/1996	Wanderer et al.	709/224
5,680,622 A	*	10/1997	Even	717/154
5,732,274 A	*	3/1998	O'Neil	717/143
5,790,863 A	*	8/1998	Simonyi	717/113
5,835,757 A	*	11/1998	Oulid-Aissa et al.	707/10
5,864,843 A	*	1/1999	Carino et al.	707/4
5,911,072 A	*	6/1999	Simonyi	717/105
6,088,731 A	*	7/2000	Kiraly et al.	709/229
6,134,709 A	*	10/2000	Pratt	717/143
6,138,098 A	*	10/2000	Shieber et al.	704/257
6,226,655 B1	*	5/2001	Borman et al.	715/501.1
6,263,339 B1	*	7/2001	Hirsch	707/102
6,282,547 B1	*	8/2001	Hirsch	707/102
6,397,263 B1	*	5/2002	Hancock et al.	709/322
6,405,209 B1	*	6/2002	Obendorf	707/103 R
6,405,365 B1	*	6/2002	Lee	717/106
6,516,356 B1	*	2/2003	Belknap et al.	719/328
6,654,747 B1	*	11/2003	Van Huben et al.	707/10
6,665,594 B1	*	12/2003	Armstrong	701/13

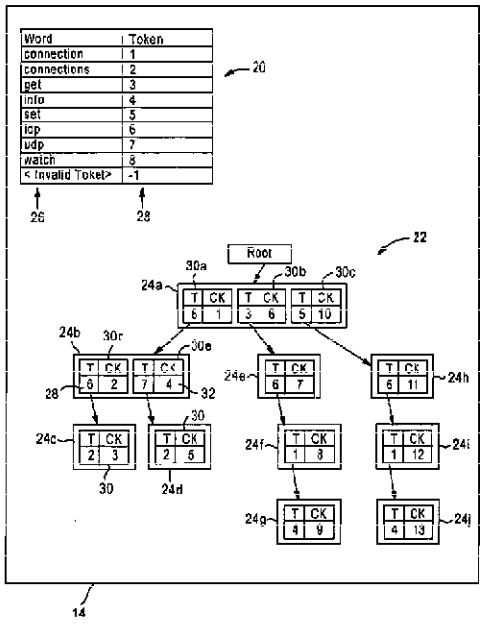
* cited by examiner

Primary Examiner—St. John Courtenay III
(74) *Attorney, Agent, or Firm* Leon R. Turkevich

(57) **ABSTRACT**

A processor based system having a parser is configured for validating a generic command received from a user relative to a command parse tree. The command parse tree includes multiple elements, each specifying at least one corresponding generic command component and a corresponding at least one command action value. The parser, upon identifying a best match among the elements, issues a prescribed command for a selected one of the management programs according to the corresponding command format based on the selected command action value. Hence, a user may control multiple management programs having respective command formats, by using a set of generic commands that are independent from the command formats, eliminating the necessity that the user needs to learn the detailed command formats and syntax.

26 Claims, 3 Drawing Sheets



Word	Token
connection	1
connections	2
get	3
info	4
set	5
icp	6
udp	7
watch	8
< Invalid Token>	-1

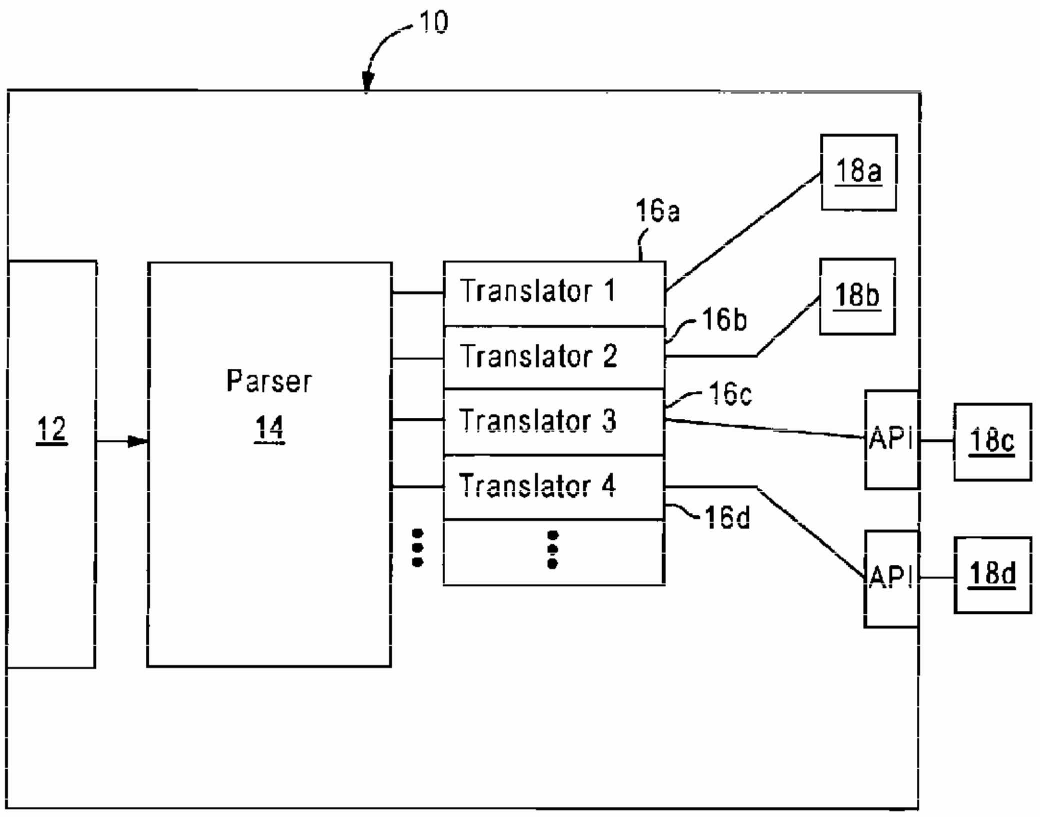
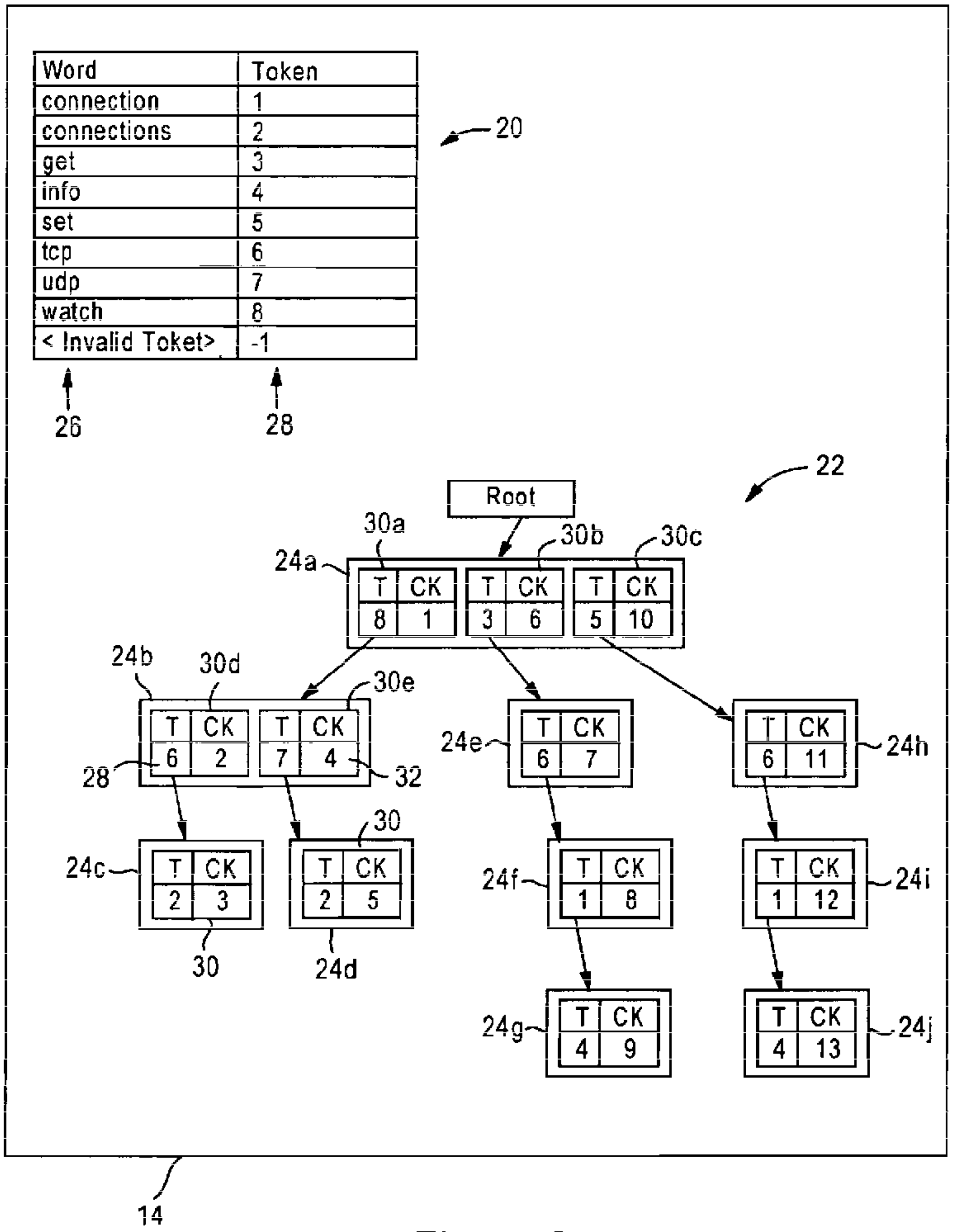

10
12
Parser
14
Translator 1
Translator 2
Translator 3
Translator 4
16a
16b
16c
16d
18a
18b
API
18c
API
18d

Figure 1



Word	Token
connection	1
connections	2
get	3
info	4
set	5
tcp	6
udp	7
watch	8
< Invalid Toket>	-1

Figure 2

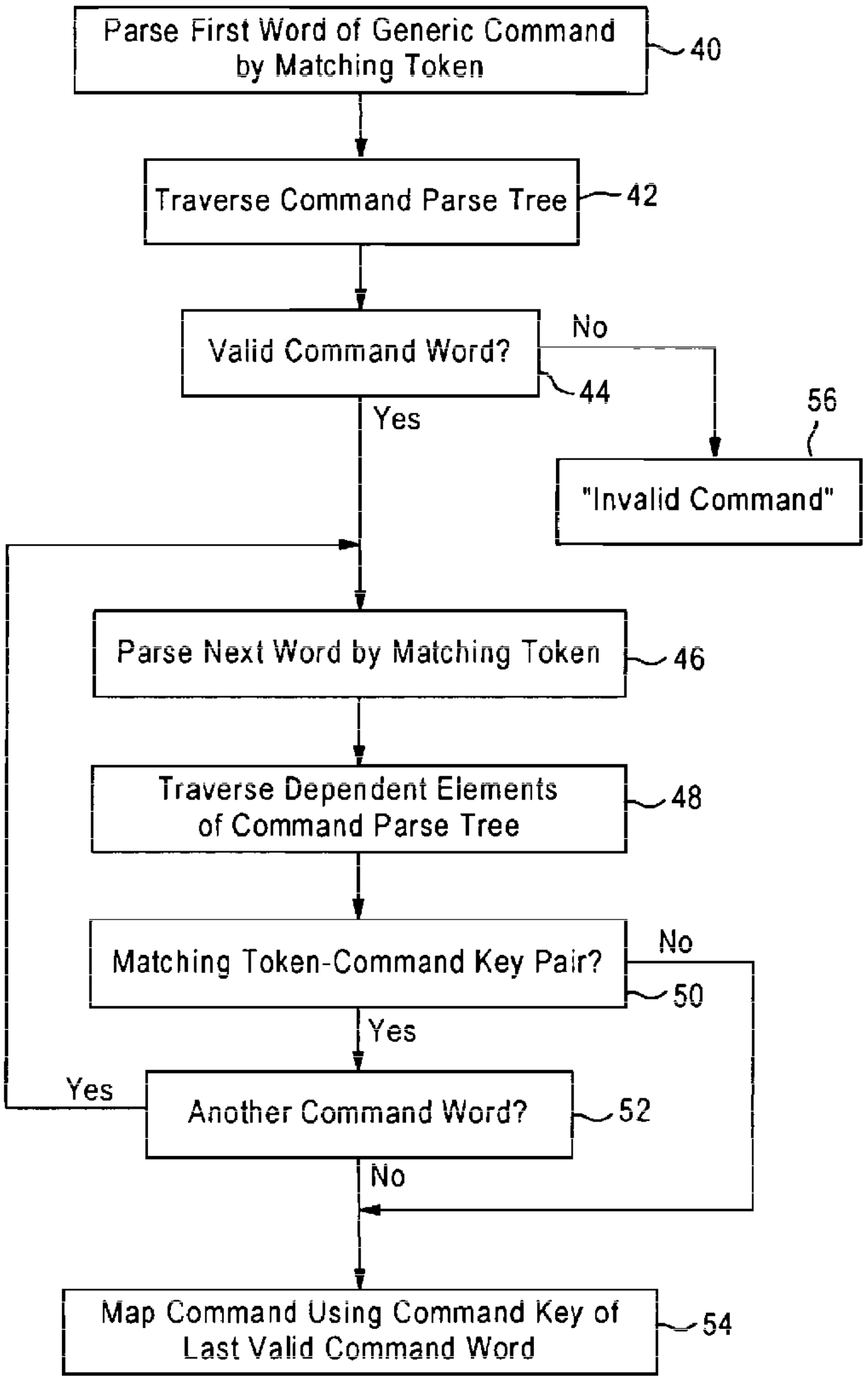

Parse First Word of Generic Command by Matching Token
40
Traverse Command Parse Tree
42
Valid Command Word?
No
44
Yes
56
"Invalid Command"
Parse Next Word by Matching Token
46
Traverse Dependent Elements of Command Parse Tree
48
Matching Token-Command Key Pair?
No
50
Yes
Yes
Another Command Word?
52
No
Map Command Using Command Key of Last Valid Command Word
54

Figure 3

GENERIC COMMAND INTERFACE FOR MULTIPLE EXECUTABLE ROUTINES

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to command and interface control of Operating Administration and Monitoring (OAM) executable routines within software systems.

2. Description of the Related Art

Operating Administration and Monitoring (OAM) tools are software-based resources used as administration and/or diagnostic tools for complex processor-based executable software systems, such as software-based unified messaging software systems. A subset of OAM tools includes Real Time Monitoring (RTM) programs, used to monitor and control selected states and processes within the software based system. For example, a given RTM program may generate a real-time display (i.e., "a screen") of selected parameters during execution of a prescribed process; the RTM program may also provide a diagnostic resource that enables resetting of various states or variables within the prescribed process. Other administration and diagnostic tools include external binary files that execute in response to a procedure call, and Simple Network Management Protocol (SNMP) agents or scripts configured for generating an e-mail message as an alarm in response to a detected event.

Hence, system administrators may attempt to utilize multiple tools within a software system in order to increase the available administration and diagnostic tools for improved system performance. The use of multiple RTM programs and other OAM tools, however, requires the users to remember the names and syntaxes of numerous commands for the respective RTM programs and OAM tools. Hence, an increase in the number of OAM tools would result in the system administrator needing to develop expertise in the command names and syntaxes for the respective OAM tools.

SUMMARY OF THE INVENTION

There is a need for an arrangement that integrates multiple RTM programs and command and control functionality for a user, without the necessity of learning the respective command formats and syntax.

There is also a need for arrangement that enables a simple command language to be utilized for control of multiple RTM programs having respective command formats.

These and other needs are attained by the present invention, where a processor based system having a parser is configured for validating a generic command received from a user relative to a command parse tree. The command parse tree includes multiple elements, each specifying at least one corresponding generic command component and a corresponding at least one command action value. The parser, upon identifying a best match among the elements, issues a prescribed command for a selected one of the management programs according to the corresponding command format based on the selected command action value. Hence, a user may control multiple management programs having respective command formats, by using a set of generic commands that are independent from the command formats, eliminating the necessity that the user needs to learn the detailed command formats and syntax.

One aspect of the present invention provides a method in a processor-based system configured for executing a plurality of management programs according to respective command formats. The method includes receiving a generic

command from the user, and validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format, the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating step including identifying one of the elements as a best match relative to the generic command. The method also includes issuing a prescribed command of a selected one of the management programs according to the corresponding command format, based on the identified one element.

Another aspect of the present invention provides a system configured for executing a plurality of management programs according to respective command formats. The system includes a parser having a command parse tree configured for validating a generic command received from a user, the command parse tree configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the parser identifying one of the elements as a best match relative to the generic command. The system also includes a plurality of translators configured for issuing commands for the management programs according to respective command formats, the parser outputting a prescribed command to a selected one of the translators based on the identified one element.

Additional advantages and novel features of the invention will be set forth in part in the description which follows and in part will become apparent to those skilled in the art upon examination of the following or may be learned by practice of the invention. The advantages of the present invention may be realized and attained by means of instrumentalities and combinations particularly pointed out in the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Reference is made to the attached drawings, wherein elements having the same reference numeral designations represent like elements throughout and wherein:

FIG. **1** is a diagram of a system configured for executing multiple management programs according to respective command formats based on a generic command set according to an embodiment of the present invention.

FIG. **2** is a diagram illustrating in detail the parser of FIG. **1** according to an embodiment of the present invention.

FIG. **3** is a diagram illustrating the validation of generic commands by the parser of FIG. 1 according to an embodiment of the present invention.

BEST MODE FOR CARRYING OUT THE INVENTION

FIG. **1** is a diagram of a system configured for executing a plurality of management programs according to respective command formats according to an embodiment of the present invention. The processor based system **10** includes a user input interface **12**, for example a terminal interface, that enables a user to input a generic command string, described below. The processor based system **10** also includes a parser **14** configured for validating the generic command received by the user input interface **12** from the user, and translators **16** configured for issuing commands to respective management programs **18** according to respective command formats. As shown in FIG. 1, the management programs **18**, implemented for example by different OAM tools such as RTM programs, may be executed within the processor based system or externally as external agents accessible using a prescribed application programming interface (API). The management programs **18** may provide different administration and maintenance functions, for example initiating various real-time screens used to monitor the internal state of executable processes within the software based system **10**; alternately, different tools **18** may allow the user to control the various states within the various component of the software based system **10** via external programs (e.g., programs **18***c* or **18***d*), or may be used to issue external alarms (e.g., SNMP manager scripts) for external routines such as message waiting indicator routines.

A disadvantage of utilizing many different tools **18** is that each tool **18** tends to have its own screen and/or command, providing difficulties for the system administrator to determine which tool is the best tool (and/or which is the best syntax) to use for a given problem.

According to the disclosed embodiment, the parser **14** and the translators **16** provide a unified administration and diagnostic tool which incorporates the functionality of all external administrative executable binary files, RTM programs, agent manipulation scripts, and various requested snapshot queries, as well as including an extensive help system. In particular, the parser **14** and the translators **16** provide a generic command syntax that integrates the functionality of the different tools **18** and that automatically selects the appropriate command for the best tool for executing a given generic command. As illustrated in Part A of the attached appendix, the new syntax provides a generic instruction set that provides an abstraction of the tool-specific command formats and syntax, enabling a user to issue command based on the relative functions, as opposed to the specific syntax for a corresponding tool **18**.

FIG. **2** is a diagram illustrating in detail the parser **14** of FIG. **1** according to an embodiment of the present invention. The parser **14** includes a command word translation table **20** and a command parse tree **22**. The command word translation table **20** is configured for storing, for each prescribed command word **26**, a corresponding token value **28** that is used by the parser **14** to identify a specific command for a selected one of the translators **16**. In particular, the command word translation table **20** includes all the command words **26** that are valid according to the generic syntax, illustrated for example in Part B of the attached appendix.

The parser **14** is configured for validating a received generic command by comparing each input command word to the command parse tree **22** to determine for the received generic command a tree element **24** identified as a best match. Each tree element **24** includes at least one token-command key pair **30** that specifies a token (T) **28** and a corresponding command key (CK) **32**, enabling the parser **14** to identify the appropriate prescribed command based on the command key specified for the matching token. In particular, the parser **14** recursively traverses the command parse tree **22** for each command word to identify the best match for the generic command. If only a portion of the generic command is identified as valid (e.g., only the first three command words are valid), the parser **14** selects the command key **32** for the matching token **28** from the last valid tree element **24**.

FIG. **3** is a diagram illustrating the method of validating a received generic command and translating the received generic command into a command for a specific management program according to an embodiment of the present invention. The operations described with respect to FIGS. **2** and **3** can be implemented as executable code that is stored

on a computer readable medium (e.g., a hard disk drive, a floppy drive, a random access memory, a read only memory, an EPROM, a compact disk, etc). The method begins in step **40**, wherein the parser begins parsing the first word of the received generic command by comparing the first input command word to the command word translation table **20** for identification of a matching token **28**. For example, assume that the parser **14** receives the valid command "watch tcp connections". The parser identifies the token value "8" as corresponding to the first command word "watch". The parser **14** than traverses the command parse tree **22** in step **42** to search for the matching token **28**. As illustrated in FIG. **2**, the parser **14** locates the matching token in the first tree element **24***a*. If the parser **14** determines in step **44** that the first command word is valid, the parser **14** continues searching the next command word in step **46**. If the first command word is invalid based on no match in the first element **24***a* of the command parse tree, the parser **14** returns an invalid command message to the user in step **56**.

The parser **14** then parses the next word (e.g., "tcp") of the received generic command in step **46** by locating the corresponding token **28** (e.g., "6" for "tcp") in the table **20**, and then traversing in step **48** the tree elements that depend from the matched tree element **24***a* (e.g., **24***b*). The parser **14** determines a match between the token **28** ("6") corresponding to the command word "tcp" in the token-command key pair **30***d* in step **50**, enabling the parser to continue for the next command word. As described above, the parser **14** repeats the process in step **52** for the third command word "connections" having the token "2" and identifying a match between the entire generic command and the token-command key **30** specified in the tree element **24***c*. The parser **14** identifies in step **54** the prescribed command for a selected one of the translators **16** based on the value of the command key **32** within the matching token-command key pair **30** (e.g., "CK=3") of the last valid command word, which maps to a translation table that specifies a specific command for a specific translator **16**.

As described above, the parser **14** can identify a command key **32** even if only a portion of the command is valid. Assume for example that the parser **14** receives the invalid command "get udp connection info". In this case, the individual command words are valid from the command word translation table **20**, however, the sequence is invalid. In particular, the command word "get" having a token value of "3" reaches the token-command key pair **30***b*, however the command word "udp" having a token value of "7" does not reach any child of the tree element **24***a*. Hence, the parser **14** uses the last valid command key ("6") in step **54** based on the matching token for the first valid word located in the token-command key pair **30***b*. The command key is mapped to a selected one of the translators **16** in an attempt to provide a command to the corresponding resource **18**. If the selected resource **18** determines that the command is invalid, the selected resource **18** at that time may prompt the user for a correct command.

The disclosed arrangement enables the use of generic commands for multiple OAM tools that have respective command syntax, resulting in a single point of entry for administering and maintaining complex software based systems. The disclosed arrangement provides the user a single set of commands and syntax to learn, facilitating the use of multiple administrative and maintenance tools.

While this invention has been described in connection with what is presently considered to be the most practical and preferred embodiment, it is to be understood that the invention is not limited to the disclosed embodiments, but, on the contrary, is intended to cover various modifications and equivalent arrangements included within the spirit and scope of the appended claims.

APPENDIX PART A: Command Syntax Mapping

Functional Item	New Syntax	Old Command Line/Syntax
Set RTM Sample Times	set watchtime <# in ms>	<Object>view -t <# in ms>
		(from command line only)
Get RTM Sample Times	get watchtime	- None -
Watch BASE Global Client/Server Information	watch acb globals	BASEview
		BASEview -g
		BASEview -h then press g
Watch BASE Thread Information	watch acb threads	BASEview -h[<Rec#>]
		BASEview then press h
Watch APP Global Information	watch acb globals	APPview
		APPview -g
		APPview -i then press g
Watch APP ICT Table Entry Information	watch acb entries	APPview -i[<Rec#>]
		APPview then press i
Watch TNT Group Information	watch cma groups	TNTview then press g
Watch TNT Session Information	watch cma sessions	TNTview
Watch H323 Information	watch h323 entries	H323view
		H323view -h | H<Rec#>
		H323view -r | -s then press h
Watch Radvision Information	watch h323 radvision	H323view -r | R<Rec#>
		H323view then press r
Watch H323 State Information	watch h323 states	H323view -s | S<Rec#>
		H323view [-r] then press s
Start BASE	start system	obs -o APP -s up
		possibly startobj.ksh
Quiesce BASE	quiesce acb	obs -o APP -s quiesce
Stop BASE	stop system	obs -o APP -s down
Start TNT	start cma	obs -o TNT -s up
Stop TNT	stop cma	obs -o TNT -s down
Quiesce TNT	quiesce cma	obs -o APP -s quiesce
Get BASE status	get system status	obs -o APP
Get Application status	watch acb entries	obs -o TNT
		ps eaf | grep <Application>
Reload all ICT entries	reload sched all	reloadsched
Reload specific ICT entry	reload sched static <Static Application> (where <Static Application> could be TNT, LogRemover, etc.)	reloadstatic <APP>
		possibly startapp <APP>
Get schedule table	watch acb entries	More SPARMHB/parms/APP/Sched*
Get current logging level for APP	set loglevels	- None -?
Set error logging level for APP	set loglevel acb error <off | local>	loglevel APP error <off | local>
Set warning logging level for APP	set loglevel acb warning <off | local>	loglevel APP warning <off | local>
Set info logging level for APP	set loglevel acb info <off | local>	loglevel APP info <off | local>
Set debug logging level for APP	set loglevel acb debug <off | local>	loglevel APP debug <off | local>
Get current logging level for TNT	get loglevels	- None - ?
Set error logging level for TNT	set loglevel cma error <off | local>	loglevel TNT error <off | local>
Set warning logging level for TNT	set loglevel cma warning <off | local>	loglevel TNT warning <off | local>
Set debug logging level for TNT	Set loglevel cma debug <off local>	loglevel TNT debug <off | local>
Get Help for RTM functions	help watch	<Object>view -?
Get Help for SNMP functions	help [start | stop | quiesce]	obs
Get Help for reload functions	help reload	reloadsched (no parms, no help)
		reloadstatic
Get Help for logging functions	help set	loglevel
Get Help for query functions	help get	Obs
Get Help on help	help help	
Exit utility	exit	Q key on RTMs

APPENDIX PART B: Generic Command Examples

Watch
This command displays the requested RTM screen
Command Usage: Watch <Object> |<Screen>|
Valid Object/Screen Pairs:

Obj	Valid Screens	Description
acb	globals	Displays ACB Global counters information
	entries	Displays ACB Entry information
	states	Displays ACB States information
	comm	Displays ACB Communication Layer information
	threads	Displays ACB Thread information
cma	groups	Display group information
	sessions	Display session information
	scr	Display CMA System Call Router information
h323	entries	Display the full H323 information screen
	radvision	Display the full radvision information screen
	states	Displays the combined (H323/Rad) states info (Note: CMA allocates double the MaxPorts configurable in H323.ini, so this screen will show double the MaxPorts entries)
sms		Displays SMS information
faxprint		Displays FaxPrint process information
notify		Displays MWI OnOT Notification info

Get
This command allows the user to request certain system variable values.
Command Usage: get <Variable>
Valid Variables

Variable	Description
watchtime	Gets the number of milliseconds between RTM (watch) screen refreshes
system status	Gets the current status of the system (up, down, quiesce)
loglevels	Gets the current run-time logging levels for ACB, CMA, and all loaded STATIC/STATIC_NOWAIT agents

Set
This command allows the used to set either UMCH variables or overall system variables
Command Usage: set <variable><value>
Valid Variable/Value Pairs:

Variable	Valid Values	Description
watchtime	Numeric in milliseconds	Sets the refresh time for RTM screens in milliseconds. Any values less than 500 will be set to 500.
loglevel acb error	"off" or "local"	Turns ACT Error level logging off or local
loglevel acb warning	"off" or "local"	Turns ACT Warning level logging off or local
loglevel acb info	"off" or "local"	Turns ACT Info level logging off or local
loglevel cma error	"off" or "local"	Turns CMA Error level logging off or local
loglevel cma warning	"off" or "local"	Turns CMA Warning level logging off or local
loglevel cma info	"off" or "local"	Turns CMA Info level logging off or local
loglevel cma debug	"off" or "local"	Turns CMA Debug level logging off or local
loglevel <PID> error	"off" or "local"	Turns <PID> Error level logging off or local (where <PID> is the PID of any scheduled agent)
loglevel <PID> warning	"off" or "local"	Turns <PID> Warning level logging off or local (where <PID> is the PID of any scheduled agent)
loglevel <PID> info	"off" or "local"	Turns <PID> Info level logging off or local (where <PID> is the PID of any scheduled agent)

-continued

APPENDIX PART B: Generic Command Examples

loglevel <PID> debug	"off" or "local"	Turns <PID> Debug level logging off or local (where <PID> is the PID of any scheduled agent)

Start
This command allows the user to start various system agents or the entire system.
Command Usage: start <Agent>
Valid Agents:

Agent	Description
system	Starts the entire system
acb	Starts a manually stopped or quiesced ACB agent
cma	Starts a manually stopped or quiesced CMA agent (new functionality coming to allow quiesce of CMA . . . start will be implemented with a spobjstate CMA R command)
logging	Starts the logging subsystem

Stop
This command allows the user to stop the TNT agent or the entire system. This command does NOT bring any running LOGSUB process down, since it is a peer process to the system and could be used by external agents which could still be running and need the service.
Command Usage: stop <Agent><Screen>
Valid Agents:

Agent	Description
system	Stops the entire system (including Logging)
acb	Stops a running or quiesced ACB agent
cma	Stops a running or quiesced CMA agent
logging	Stops the logging subsystem only if the ACB agent is down

Quiesce
This command allows the user to quiesce the APP or TNT agent.
Command Usage: quiesce <Agent>
Valid Agents:

Agent	Description
acb	Quiesces acb
cma	Quiesces a running CMA agent

Reload
This command allows the user to reload various configuration files.
Command Usage: reload <Agent> [<Parameters>]
Valid Applications:

Application	Parameters	Description
cmaloglevels		Causes CMA to reload the DBG and TRACE sections of the SPARMLIB/TNT/parms/TNT.ini configuration file.
Dialmap		Causes CMA to reload $PARMLIB/TNT/parms/DialMap.ini
Route	"acb" or "cma"	Causes ACB to reload either $PARMLIB/parms/APP/Route.<hostname> (acb) or $PARMLIB/parms/TNT/Route.<hostname> (cma)
Sched	"all" or "static <token>"	Causes ACB to reschedule either all or a single agent defined in SPARMLIB/parms/APP/Schedule.<hostname>. (Use of the static <token> parameter requires that <token> had previously been scheduled either as a STATIC or STATIC_NOWAIT.)

Help
This command allows the user to get help on the valid commands available, their usage, and what they mean.
Command Usage: help [<command>]
Valid Commands:

Commands	Description

-continued

APPENDIX PART B: Generic Command Examples

<None>	Help without any parameters gives the users either a list of the top level commands (help set to "short") or the top level commands and all valid sub commands under each top level commands (help set to "full")
watch	Gives the user a list of valid screens and what each one is
get	Gives the user a list of valid variables to query
set	Gives the user a list of valid variables and valid values for each variable
start	Gives the user a list of valid Agents to start
stop	Gives the user a list of valid Agents to stop
quiesce	Gives the user a list of valid Agents to quiesce
reload	Gives the user a list of valid Configurables (and possibly optional parameters) to reload.
help	The ubiquitous help on help (probably not necessary . . .)

What is claimed is:

1. A method in a processor-based system configured for executing a plurality of management programs according to respective command formats, the method comprising:

receiving a generic command from the user;

validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format, the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating step including identifying one of the elements as a best match relative to the generic command; and

issuing a prescribed command of a selected one of the management programs according to the corresponding command format, based on the identified one element.

2. The method of claim **1**, wherein the generic command includes at least one input command word, the validating step including:

comparing each input command word to a command word translation table, configured for storing for each prescribed command word a corresponding token, for identification of a matching token; and

determining a presence of the matching token within the command parse tree for each input command word.

3. The method of claim **2**, wherein the determining step includes recursively traversing the command parse tree based on an order of the input command words for identification of the matching token within the identified one element.

4. The method of claim **3**, wherein the issuing step includes issuing the prescribed command based on a corresponding command key specified for the matching token within the identified one element.

5. The method of claim **4**, wherein the issuing step further includes accessing a prescribed translator configured for converting the generic command according to the corresponding command format into the prescribed command based on the corresponding command key.

6. The method of claim **5**, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command.

7. The method of claim **6**, further comprising executing the prescribed command within the corresponding selected one management program.

8. The method of claim **1**, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command.

9. The method of claim **8**, further comprising executing the prescribed command within the corresponding selected one management program.

10. A system configured for executing a plurality of management programs according to respective command formats, the system comprising:

a parser having a command parse tree configured for validating a generic command received from a user, the command parse tree configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the parser identifying one of the elements as a best match relative to the generic command; and

a plurality of translators configured for issuing commands for the management programs according to respective command formats, the parser outputting a prescribed command to a selected one of the translators based on the identified one element.

11. The system of claim **10**, wherein the parser further comprises a command word translation table configured for storing for each prescribed command word a corresponding token for identification of a matching token, the parser configured for determining a presence of the matching token within the command parse tree for each input command word.

12. The system of claim **11**, wherein the parser recursively traverses the command parse tree based on an order of the input command words for identification of the matching token within the identified one element.

13. The system of claim **12**, wherein the parser validates at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command.

14. A computer readable medium having stored thereon sequences of instructions for executing a plurality of management programs according to respective command formats, the sequences of instructions including instructions for performing the steps of:

receiving a generic command from the user;

validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format, the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating step including identifying one of the elements as a best match relative to the generic command; and

issuing a prescribed command of a selected one of the management programs according to the corresponding command format, based on the identified one element.

15. The medium of claim **14**, wherein the generic command includes at least one input command word, the validating step including:

comparing each input command word to a command word translation table, configured for storing for each prescribed command word a corresponding token, for identification of a matching token; and determining a presence of the matching token within the command parse tree for each input command word.

16. The medium of claim **15**, wherein the determining step includes recursively traversing the command parse tree based on an order of the input command words for identification of the matching token within the identified one element.

17. The medium of claim **16**, wherein the issuing step includes issuing the prescribed command based on a corresponding command key specified for the matching token within the identified one element.

18. The medium of claim **17**, wherein the issuing step further includes accessing a prescribed translator configured for converting the generic command according to the corresponding command format into the prescribed command based on the corresponding command key.

19. The medium of claim **18**, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command.

20. The medium of claim **19**, further comprising instructions for performing the step of executing the prescribed command within the corresponding selected one management program.

21. The medium of claim **14**, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command.

22. The medium of claim **21**, further comprising instructions for performing the step of executing the prescribed command within the corresponding selected one management program.

23. A system configured for executing a plurality of management programs according to respective command formats, the system comprising:

means for validating a generic command received from a user, the validating means configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value, the validating means identifying one of the elements as a best match relative to the generic command; and

a plurality of translators configured for issuing commands for the management programs according to respective command formats, the validating means outputting a prescribed command to a selected one of the translators based on the identified one element.

24. The system of claim **23**, wherein the validating means comprises a command word translation table configured for storing for each prescribed command word a corresponding token for identification of a matching token, the validating means configured for determining a presence of the matching token for each input command word.

25. The system of claim **24**, wherein the validating means recursively validates each input command word based on an order of the input command words for identification of the matching token within the identified one element.

26. The system of claim **25**, wherein the validating means validates at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command.

* * * * *

EXHIBIT E

US007953886B2

(12) **United States Patent**
Bansal et al.

(10) **Patent No.: US 7,953,886 B2**
(45) **Date of Patent: May 31, 2011**

(54) **METHOD AND SYSTEM OF RECEIVING AND TRANSLATING CLI COMMAND DATA WITHIN A ROUTING SYSTEM**

(75) Inventors: **Anil Bansal**, Fremont, CA (US); **Jung Tjong**, Sunnyvale, CA (US); **Prakash Bettadapur**, San Jose, CA (US); **Sastry Varanasi**, Sunnyvale, CA (US)

(73) Assignee: **Cisco Technology, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 693 days.

(21) Appl. No.: **11/178,136**

(22) Filed: **Jul. 8, 2005**

(65) **Prior Publication Data**

US 2007/0011348 A1 Jan. 11, 2007

(51) **Int. Cl.**
G06F 15/173 (2006.01)
(52) **U.S. Cl.** **709/238**
(58) **Field of Classification Search** 709/223 226, 709/238
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,778,223 A		7/1998	Velissaropoulos et al.	
6,278,455 B1		8/2001	Baker	
6,457,173 B1		9/2002	Gupta et al.	
6,553,366 B1		4/2003	Miller et al.	
6,675,370 B1		1/2004	Sundaresan	
6,697,967 B1	*	2/2004	Robertson	714/43
6,724,409 B1		4/2004	Maddocks	
6,738,781 B1		5/2004	Mustoe	
6,744,433 B1		6/2004	Bastos et al.	
6,907,572 B2	*	6/2005	Little et al.	715/762
6,954,790 B2		10/2005	Forslow	
6,959,329 B2	*	10/2005	Thakor	709/220
6,959,332 B1		10/2005	Zavalkovsky et al.	
7,054,901 B2		5/2006	Shafer	709/203
7,054,924 B1	*	5/2006	Harvey et al.	709/220
7,058,699 B1		6/2006	Chiou et al.	
7,072,946 B2		7/2006	Shafer	709/217
7,072,985 B1	*	7/2006	Lev-Ami et al.	709/246
7,149,738 B2	*	12/2006	Kumar et al.	707/9
7,200,548 B2	*	4/2007	Courtney	703/27
7,321,929 B2		1/2008	Rolfe	
2002/0198974 A1		12/2002	Shafer	
2003/0033589 A1		2/2003	Reyna	
2003/0046370 A1		3/2003	Courtney	
2003/0048287 A1	*	3/2003	Little et al.	345/705
2003/0051008 A1	*	3/2003	Gorthy et al.	709/220

(Continued)

OTHER PUBLICATIONS

Sun Management Center 3.6.1 User's Guide (Chapter 20 only); Sun Microsystems; May 2006.*

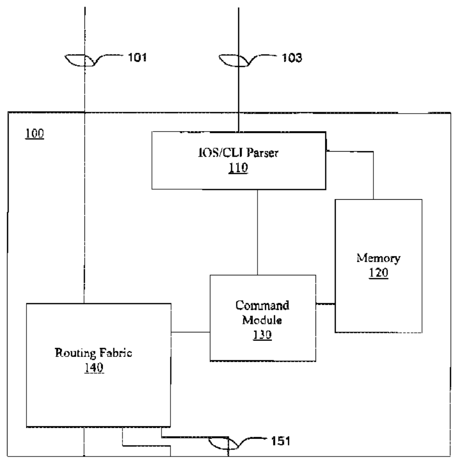
(Continued)

Primary Examiner — Kenny S Lin
Assistant Examiner — Farhad Ali
(74) *Attorney, Agent, or Firm* — Stolowitz Ford Cowger LLP

(57) **ABSTRACT**

A method and system of receiving and translating data using an internetwork operating system (IOS) command line interface (CLI) parser subsystem of a routing system are shown. Input is received at the IOS CLI parser subsystem. The input is traversed. Where the input originates from outside of the router, the input is translated into a corresponding CLI statement. Where the input originates from within the router, the input is translated into a prescribed output format. The translated input is output.

15 Claims, 4 Drawing Sheets



Routing System 100

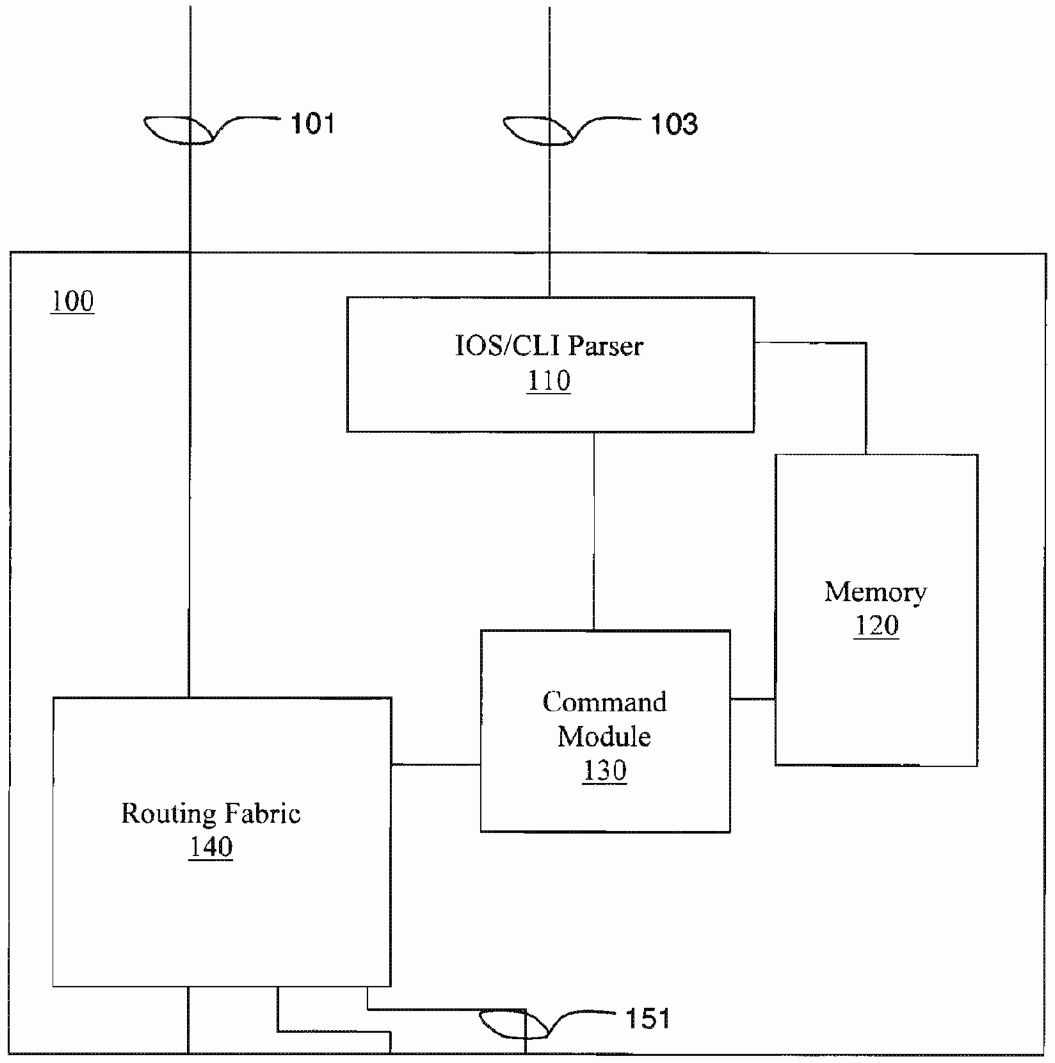
U.S. PATENT DOCUMENTS

2003/0101240	A1 *	5/2003	Courtney	709/220
2004/0040016	A1 *	2/2004	Pearce et al.	717/141
2004/0078457	A1 *	4/2004	Tindal	709/223
2004/0090439	A1	5/2004	Dillner	
2004/0117452	A1 *	6/2004	Lee et al.	709/208
2004/0168124	A1	8/2004	Beisiegel et al.	
2004/0205562	A1 *	10/2004	Rozek et al.	715/513
2005/0021502	A1	1/2005	Chen et al.	
2005/0091068	A1 *	4/2005	Ramamoorthy et al.	705/1
2006/0031427	A1 *	2/2006	Jain et al.	709/220
2006/0080425	A1 *	4/2006	Wood et al.	709/223
2006/0129980	A1 *	6/2006	Schmidt et al.	717/114
2006/0230378	A1	10/2006	Waddington et al.	
2006/0242403	A1	10/2006	Joshi et al.	

OTHER PUBLICATIONS

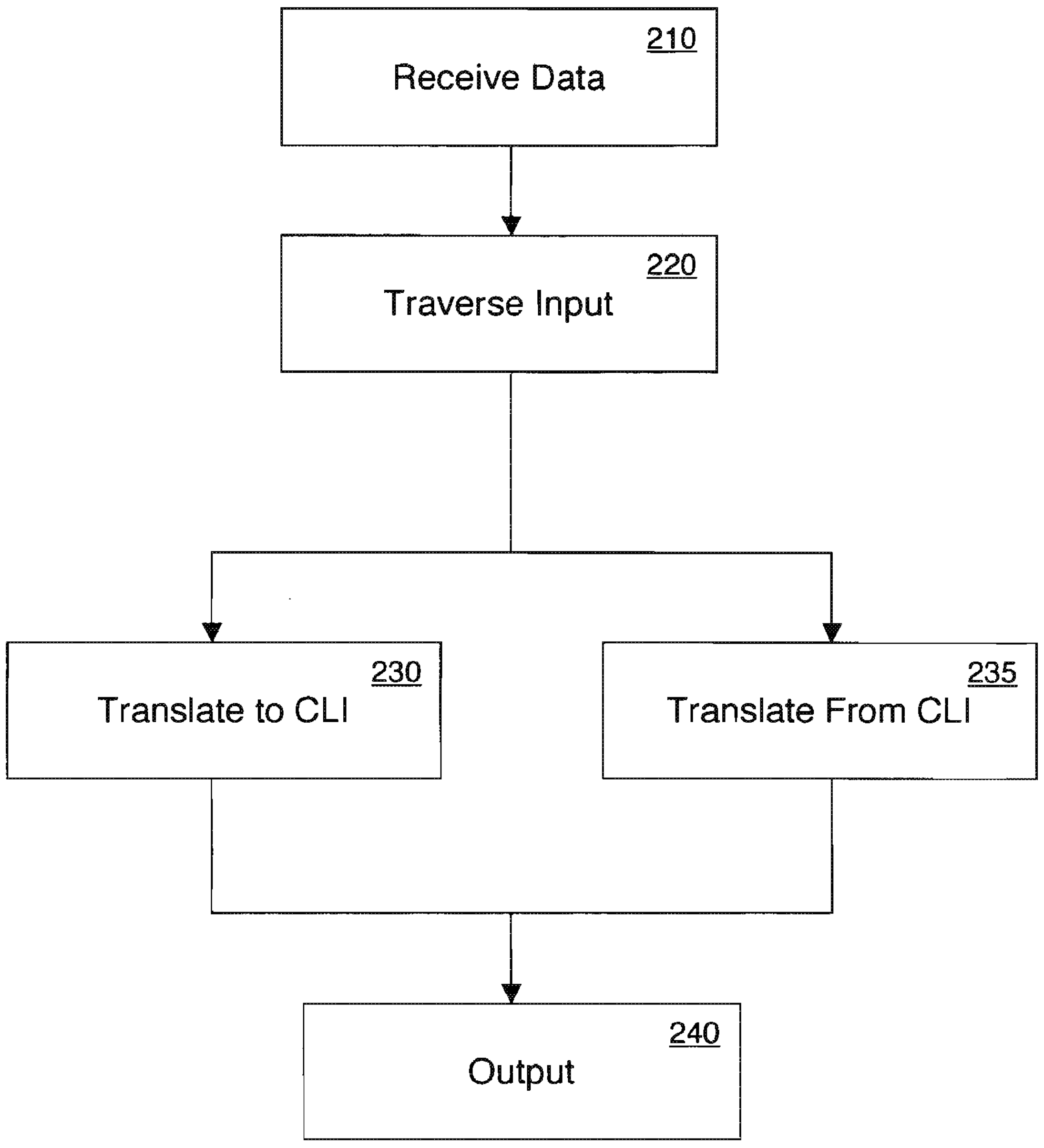
Using the Command-Line Interface in Cisco IOS and Cisco IOS XE Software; Cisco Systems; Last Updated Mar. 5, 2009.*
Network Appliance, Inc.; Netcache Command Line Interface; 1999-2000; www.billfernandez.com/bfdz/portfolio/netcache_cli/index.htm; 2 Pages.
Allegro Software Development Corp.; Rom CLE Embedded Command Line Interface (CLI) Toolkits; www.allegrosoft.com/romcli.html; 3 Pges.
Matuska, Miroslav; IOAConvert: IOS to XML Router Configuration File Converter—Command Reference, CESNET Technical Report No. 30/2003; Oct. 12, 2003; 36 Pages.
Stolowitz Ford Cowger LLP, Related Case Listing; Nov. 11, 2009; 1 Page.

* cited by examiner



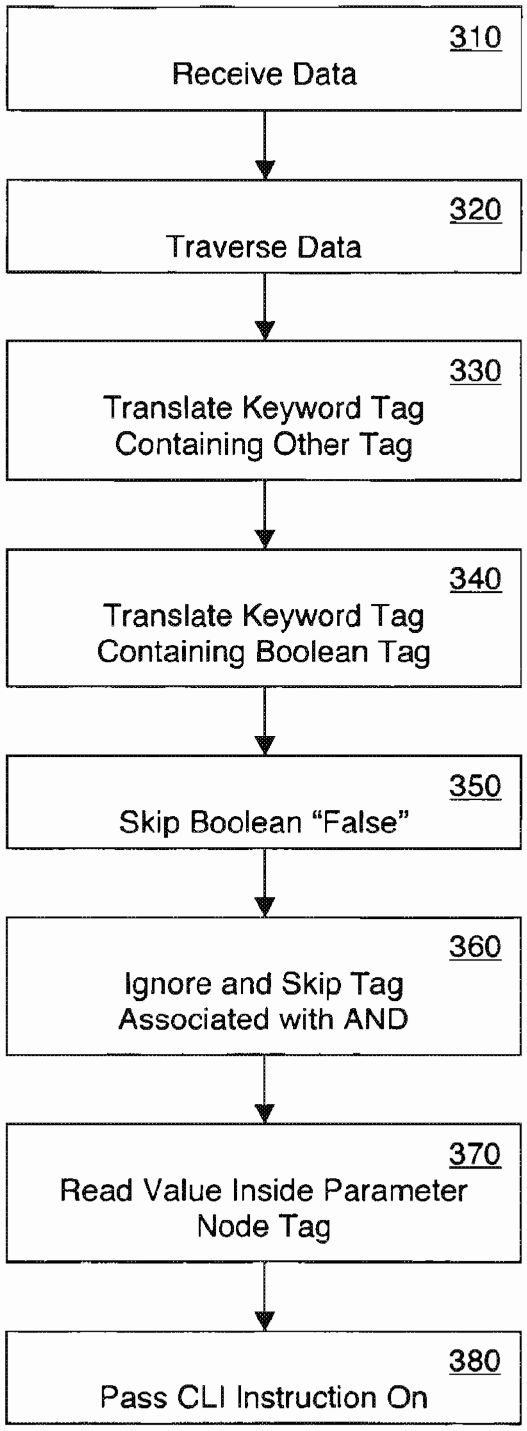
Routing System 100

Figure 1



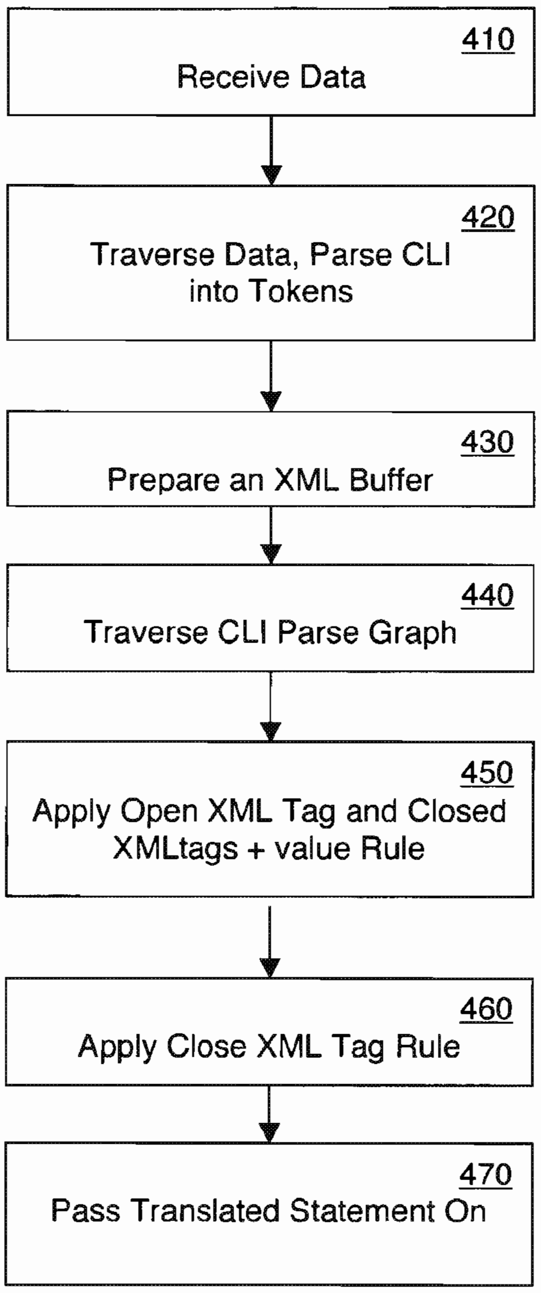
Flowchart 200

Figure 2


310
Receive Data
320
Traverse Data
330
Translate Keyword Tag Containing Other Tag
340
Translate Keyword Tag Containing Boolean Tag
350
Skip Boolean "False"
360
Ignore and Skip Tag Associated with AND
370
Read Value Inside Parameter Node Tag
380
Pass CLI Instruction On

Flowchart 300

Figure 3


410
Receive Data
420
Traverse Data, Parse CLI into Tokens
430
Prepare an XML Buffer
440
Traverse CLI Parse Graph
450
Apply Open XML Tag and Closed XMLtags + value Rule
460
Apply Close XML Tag Rule
470
Pass Translated Statement On

Flowchart 400

Figure 4

METHOD AND SYSTEM OF RECEIVING AND TRANSLATING CLI COMMAND DATA WITHIN A ROUTING SYSTEM

BACKGROUND

1. Field of the Invention

The present invention relates to routing systems for computer networks, and more particularly to the transmission of instructions to and receipt of data from such routing systems.

2. Related Art

Access and configuration of a routing system involves sending commands and instructions to and receiving information from the router itself. For routers using a version of the internetwork operating system (IOS), access is accomplished through the use of the IOS command line interface (CLI). IOS CLI is a comprehensive interface, which has expanded continuously as technology has improved over the past twenty years. Many companies now strive to support some variation on IOS CLI in their routing systems, and many consumers have invested heavily in IOS CLI support, developing complicated scripts to handle various configuration and access needs. As such, it is desirable for any improvements to router access and control to acknowledge the existing investments of consumers.

IOS CLI is not the most program-friendly of interfaces, however. Twenty years of consistency and backwards-compatibility, when coupled with consistent improvements to the hardware and implementation of new features, has created an extensive interface. While a human user of IOS CLI may be able to sort through the complicated input and output scheme to input information and extract important data, it has proven to be a very difficult and cumbersome task to automate.

A system and/or method that allows for an easy, more structured approach to accessing and configuring a router, while still making use of the significant advantages and experience associated with IOS CLI, would be advantageous.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** depicts a block diagram of an exemplary routing system, in accordance with one embodiment of the present invention.

FIG. **2** is a flowchart of a method for receiving and translating data using an internetwork operating system (IOS) command line interface (CLI) parser subsystem of a routing system, in accordance with one embodiment of the present invention.

FIG. **3** is a flowchart of a method for receiving XML statements into an IOS CLI parser subsystem of a routing system and translating the XML statements into CLI commands, in accordance with one embodiment of the present invention.

FIG. **4** is a flowchart of a method for receiving CLI input into an IOS CLI parser subsystem of a routing system and translating the input into XML statements, in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION

A method of receiving and translating data using an internetwork operating system (IOS) command line interface (CLI) parser subsystem of a routing system, and a routing system incorporating that method, are disclosed. Reference will now be made in detail to several embodiments of the invention. While the invention will be described in conjunction with the alternative embodiment(s), it will be understood

that they are not intended to limit the invention to these embodiments. On the contrary, the invention is intended to cover alternative, modifications, and equivalents, which may be included within the spirit and scope of the invention as defined by the appended claims.

Furthermore, in the following detailed description of the present invention, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be recognized by one skilled in the art that the present invention may be practiced without these specific details or with equivalents thereof. In other instances, well-known methods, procedures, components, and circuits have not been described in detail as not to unnecessarily obscure aspects of the present invention.

NOTATION AND NOMENCLATURE

Some portions of the detailed descriptions, which follow, are presented in terms of procedures, steps, logic blocks, processing, and other symbolic representations of operations on data bits that can be performed on computer memory, such as a computer-usable medium. These descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. A procedure, computer-executed step, logic block, process, etc., is here, and generally, conceived to be a self-consistent sequence of steps or instructions leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated in a computer system. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussions utilizing terms such as "accessing," "writing," "including," "testing," "using," "traversing," "associating," "identifying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

With reference now to FIG. **1**, a block diagram of a routing system **100** is depicted, in accordance with one embodiment of the invention. It is appreciated that the present invention is readily implemented in a variety of routing and/or switching configurations. Routing system **100** includes IOS/CLI Parser **110**, Memory **120**, Command Module **130**, Routing Fabric **140**, input port **101**, programming port **103**, and several output ports **151**. Command Module **130** is coupled to IOS/CLI Parser **110**, Memory **120**, and Command Module **130**. It is appreciated that the present invention is readily adaptable for utilization in systems of various configurations, including systems with varying numbers of ports. In one embodiment, input port **101** and programming port **103** might be combined into a single port. Moreover, differing types of input ports are used in different embodiments: direct physical terminal console ports are used in one embodiment for interactive human users; another embodiment supports the use of remote terminal protocols, such as SSH and telnet, for remote administration; a third embodiment allows for receipt of input as a batch process. No one type of input port is preferred for the utilization of this invention. One embodiment includes three output ports **151**; another embodiment might use differing quantities. Similarly, another embodiment of a routing system may used separate memory for the IOC CLI parser and the processor.

The components of routing system **100** cooperatively operate to route or otherwise distribute signals received, pursuant to instructions received from a user. A user need not be an actual person; command statements may be issued automatically by an outside program, which would be considered a "user" as well. It is appreciated that the term "command statements" is intended to be exemplary only; other types of statements, such as requests for information, are also received by routing system **100** and are treated in similar fashion. In one embodiment, such command statements are received by routing system **100** through programming port **103**, and are passed to IOS/CLI Parser **110**. In one embodiment of the present invention, these command statements may be formatted in accordance with the CLI rules and behaviors expected by IOS/CLI Parser **110**, or in accordance with an XML schema of the CLI rules and behaviors. In another embodiment, these command statements can be formatted in accordance with another set of rules and behaviors, or presented in a language other than XML or CLI. Once received by IOS/CLI Parser **110**, the command statements are parsed according to instructions programmed into IOS/CLI Parser **110**. Alternatively, the command statements can be parsed in accordance with a model of the CLI command structure stored in Memory **120**. Non-CLI command statements must also be translated into CLI statements that Command Module **130** can interpret.

Once the command statements have been parsed and translated, according to one embodiment, they are passed to Command Module **130**. Command Module **130** acts upon the received command statements in accordance with instructions and data stored in Memory **120**. When the command statements call for a response from routing system **100**, such as an acknowledgement of an altered setting or information regarding the performance of routing system **100**, a response statement is returned to IOS/CLI Parser **110**.

In one embodiment, such a response statement would be received from Command Module **130** in CLI. IOS/CLI Parser **110** would translate the CLI response into an XML response in accordance with an XML schema of the CLI rules and behaviors. In other embodiments, a response statement could be translated in accordance with a different set of rules or behaviors, or outputted in languages other than XML.

Command Module **130** receives ingress or input information on through input port **101**, and routes it to an appropriate output port **151**. The information flow through Command Module **130** is directed by Command Module **130** based upon directions stored in Memory **120** or received through programming port **103**, as detailed above. p**103** also directs scheduling of information flows through routing system **100**.

With reference now to FIG. **2**, a flowchart **200** of a method for receiving and translating data using an IOS CLI parser subsystem of a routing system is depicted, in accordance with one embodiment of the invention. Although specific steps are disclosed in flowchart **200**, such steps are exemplary. That is, embodiments of the present invention are well suited to performing various other (additional) steps or variations of the steps recited in flowchart **200**. It is appreciated that the steps in flowchart **200** may be performed in an order different than presented, and that not all of the steps in flowchart **200** may be performed.

In step **210** of flowchart **200**, in one embodiment, input data is received at the IOS/CLI Parser **110** of a routing system **100**. In one embodiment, this input may originate from outside the routing system **100**, and be passed to the IOS/CLI Parser **110** by way of programming port **103**. In another embodiment, the input may originate from inside the routing system **100**, and be passed to the IOS/CLI Parser **110** from Command Module **130**; in one embodiment, such input is a representation of the current configuration settings of routing system **100**. This input data can take the form of CLI statements. Alternatively, the input can be formatted in accordance with another language syntax; one embodiment calls for input to be formatted in accordance with a specific XML schema of the CLI syntax.

In step **220** of flowchart **200**, in one embodiment, the input received in step **210** is traversed. During traversal, the source of the data can be ascertained (e.g. whether the input originated from inside or outside routing system **100**). This determination affects what operations are performed on the input, as is explained in greater depth with reference to FIGS. **3** and **4**, below.

In step **230** of flowchart **200**, in one embodiment, when the input received in step **210** originated from outside of routing system **100**, the input is translated into CLI statements. According to one embodiment, the input was originally formatted according to an XML schema of the CLI rules and behaviors. In other embodiments, the input might be received in a different language and translated into CLI. Greater explanation of this transformative behavior is explained with reference to FIG. **3**, below.

In step **235** of flowchart **200**, in one embodiment, when the input received in step **210** originated from inside or routing system **100**, the input is translated from CLI statements into a different format. According to one embodiment, the CLI statements would be translated into corresponding XML statements, in accordance with an XML schema of the CLI rules and behaviors. In other embodiments, the input would be translated into other desirable output formats. Greater explanation of this transformative behavior is explained with reference to FIG. **4**, below.

In step **240** of flowchart **200**, in one embodiment, the output of steps **230** or **235** is passed along. In the case of step **230**, the transformed input is passed to Command Module **130** within routing system **100** for further action, now that it is in CLI format. In the case of step **235**, the transformed input is passed to programming port **103**, where it leaves routing system **100**.

With reference now to FIG. **3**, a flowchart **300** of a method for receiving XML statements into an IOS CLI parser subsystem of a routing system and translating the XML statements into CLI commands, in accordance with one embodiment of the present invention. Although specific steps are disclosed in flowchart **300**, such steps are exemplary. That is, embodiments of the present invention are well suited to performing various other (additional) steps or variations of the steps recited in flowchart **300**. It is appreciated that the steps in flowchart **300** may be performed in an order different than presented, and that not all of the steps in flowchart **300** may be performed.

In step **310** of flowchart **300**, in one embodiment, input data is received at the IOS/CLI Parser **110** of a routing system **100**. In this embodiment, this input originates from outside the routing system **100**, and is passed to the IOS/CLI Parser **110** by way of programming port **103**. In this embodiment, the input is formatted in accordance with a specific XML schema of the CLI syntax. This schema minimizes the translation required from XML to CLI by requiring all existing CLI rules and behaviors to be followed by the XML schema, e.g. the sequencing of commands within an XML request must follow CLI configuration rules. XML tags consist of CLI keywords, with some additional tags for command parameters. An example of input formatted in accordance with one XML schema is presented in Table 1, below.

TABLE 1

```
<add>
<k__mpls__label>
<k   range>
<range-min>10</range-min>
      <range-max>300</range-max>
<k__static>
      <static-min>30< static-min>
      <static-max>150</static-max>
</k__static>
</k   range>
 </k__mpls__label>
</add>
```

In step **320** of flowchart **300**, in one embodiment, the input received in step **310** is traversed. In this embodiment, where the input is formatted in accordance with a specific XML schema of the CLI syntax, this step includes traversing the XML fragments associated with each XML Command from top to bottom.

In step **330** of flowchart **300**, in one embodiment, each XML tag has the following rule applied to it. If an XML keyword tag contains another keyword, it is translated into one or more CLI keywords. Multiple CLI keywords may have been concatenated into a single XML tag delimited by a character, such as the underscore character "_"; in such cases, the keywords are extracted from the tag and translated into multiple CLI keywords.

In step **340** of flowchart **300**, in one embodiment, each XML tag has the following rule applied to it. If an XML keyword tag contains the boolean value "true", it is translated into the associated CLI keyword.

In step **350** of flowchart **300**, in one embodiment, each XML tag has the following rule applied to it. If an XML keyword tag contains the boolean valude "false", it is skipped.

In step **360** of flowchart **300**, in one embodiment, each XML tag has the following rule applied to it. If an XML keyword tag is associated with an AND node, it is ignored and skipped; such tags are identifiable through naming conventions.

In step **370** of flowchart **300**, in one embodiment, each XML tag has the following rule applied to it. If an XML keyword tag is a parameter node tag, the values inside the tag are extracted.

In step **380** of flowchart **300**, in one embodiment, the translated CLI command statement is passed on. In one embodiment, the translated CLI command statement is passed to command module **130** to be acted upon in accordance with the rules and behaviors associated with CLI. An example of a translated CLI command is presented below, in Table 2. This is the result of translating the example input from Table 1, above, into CLI.

TABLE 2

```
mpls label range 10 300 static 30 150
```

With reference now to FIG. **4**, a flowchart **400** of a method for receiving CLI statements into an IOS CLI parser subsystem of a routing system and translating the CLI statements into XML statements, in accordance with one embodiment of the present invention. Although specific steps are disclosed in flowchart **400**, such steps are exemplary. That is, embodiments of the present invention are well suited to performing various other (additional) steps or variations of the steps recited in flowchart **400**. It is appreciated that the steps in flowchart **400** may be performed in an order different than presented, and that not all of the steps in flowchart **400** may be performed.

In step **410** of flowchart **400**, in one embodiment, input data is received at the IOS/CLI Parser **110** of a routing system **100**. In this embodiment, this input originates from inside the routing system **100**, and is passed to the IOS/CLI Parser **110** from command module **130**. In this embodiment, the input is formatted in accordance with the CLI syntax, and is to be translated into a specific XML schema of the CLI syntax. This schema minimizes the translation required from CLI to XML by requiring all existing CLI rules and behaviors to be followed by the XML schema, e.g. the sequencing of commands within an XML request must follow CLI configuration rules. XML tags consist of CLI keywords, with some additional tags for command parameters.

In step **420** of flowchart **400**, in one embodiment, the input received in step **410** is traversed. The CLI statements are parsed into tokens.

In step **430** of flowchart **400**, in one embodiment, the IOS CLI parser also prepares an XML buffer in memory **120**.

In step **440** of flowchart **400**, in one embodiment, each CLI token has the following rule applied to it. The CLI parse graph stored in memory **120** is traversed, with respect to each token. In this embodiment, the parse graph has been modified slightly from preceding implementations, to add additional information to IOS parse node types that represent CLI keywords and parameters. In the case of keyword nodes, the following information is added: "parent_label," which is a label given to AND nodes; "is_boolean," which is true if the keyword is a boolean value; and "has_more_tag," which is true if the keyword is to be merged with the next keyword. In the case of parameter nodes, the following is added: "label," the label given to parameter nodes; and "parent_label," which is a label given to AND nodes.

In step **450** of flowchart **400**, in one embodiment, each CLI token has the following rule applied to it. The "Open XML Tag and Closed XML tags+Value" rule, presented below in Table 3, assembles an XML statement from modified CLI tokens, placing the tokens in the XML buffer and adding characters as required to conform to the XML schema implemented.

TABLE 3

```
    Open XML Tag and Closed XML tags + Value Rule
Open XML Tag is introduced by:
    Non-boolean value keyword parse node
    Keyword or parameter node's parent   label
For Parameter node:
    Add "<" to XML buffer
    Add parameter node's label string to XML buffer
    Add ">" to XML buffer
    Add parameter node's value to XML buffer
    Add "</" to XML buffer
    Add parameter node's label string to XML buffer
    Add ">" to XML buffer
For boolean keyword node (is__boolean is true)
    Add "<" to XML buffer
    Add keyword string to XML buffer
    Add ">true</" to XML buffer
    Add keyword string to XML buffer
```

TABLE 3-continued

```
        Add ">" to XML buffer
    For non-boolean keyword node
        if (last character in buffer is not "   ") {
            Add "_" to XML buffer
        } else (keyword's has   more   tag is false) {
            Add ">" to XML buffer
        }
```

In step **460** of flowchart **400**, in one embodiment, each CLI token has the following rule applied to it. The "Close XML tag" rule is used to determine when to close an open XML tag to stop the nesting and move back up the hierarchy. In this embodiment, the rule has two conditions, either of which may be satisfied: reaching the End of Line (EOL), or the token's "parent_label" is different from the preceding node's "parent_label" will close an open XML tag.

In step **470** of flowchart **400**, in one embodiment, the translated XML statement will leave IOS CLI parser **110**. In one embodiment, the translated XML statement will be returned to the user via programming port **103**.

The foregoing descriptions of specific embodiments of the present invention have been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed, and obviously many modifications and variations are possible in light of the above teaching. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the Claims appended hereto and their equivalents.

What is claimed is:

1. A method comprising:

receiving, with a command line interface (CLI) parser, an input command configured to request an operation be performed by a routing system, wherein the input command is configured in an extensible markup language (XML) format having a CLI syntax with CLI keywords sequenced according to configuration rules for CLI commands;

translating, with the CLI parser, the input command from the XML format having the CLI syntax into a CLI command that, when executed, is configured to prompt the routing system to perform the operation, wherein the translating of the input command into the CLI command includes identifying at least one XML tag that includes an XML parameter to indicate the XML tag includes one or more CLI keywords, extracting the one or more CLI keywords from the input command, and arranging the one or more CLI keywords into the CLI command according to the CLI syntax of the input command, wherein the routing system is configured to perform the operation responsive to the execution of the CLI command;

translating an output message, generated in response to performance of the operation, from a CLI format into an XML format having the CLI syntax, wherein the translating includes parsing the output message to identify at least one CLI token, translating each CLI token of the output message into a corresponding XML value according to a stored mapping of CLI tokens-to-XML values, and generating the output message in the XML format with the XML values; and

transmitting the output message in the XML format having the CLI syntax to a remote device external from the routing system.

2. The method of claim **1**, wherein the input command is formatted in accordance with an XML schema of CLI rules and behaviors enforced by an internetwork operating system (IOS) command line interface (CLI) parser subsystem.

3. The method of claim **2**, wherein the translation of the input command from XML format having a CLI syntax into a CLI command comprises:

parsing the input command to identify an XML command attribute;

traversing the input after the XML command attribute to identify any keywords and any parameters associated with the XML command attribute;

translating the XML command attribute into the CLI command; and

translating the keywords and any parameters into associated attributes of the CLI command.

4. The method of claim **1**, wherein the output message in the XML format having the CLI syntax includes data formatted in accordance with an XML data model of CLI rules and behaviors enforced by an internetwork operating system (IOS) command line interface (CLI) parser subsystem.

5. The method of claim **1**, wherein the translation of the output message from the CLI format into the XML format having the CLI syntax comprises:

parsing the output message to identify at least one CLI token;

accessing a stored mapping of CLI tokens-to-XML values;

translating each CLI token of the output message into a corresponding XML value, in accordance with said stored mapping; and

generating the output message in the XML format having the CLI syntax with the XML values.

6. A computer-usable memory device having computer-readable program code embedded therein for causing a computer system to:

receive an input command requesting an operation be performed by a routing system, wherein the input command is configured in an extensible markup language (XML) format having a command line interface (CLI) syntax with CLI keywords sequenced according to configuration rules for CLI commands;

translate the input command from the XML format having the CLI syntax into a CLI command, wherein the routing system is configured to execute the CLI command and perform the operation, and wherein the computer system is further configured to translate the input command by identifying at least one XML tag that includes an XML parameter to indicate the XML tag includes one or more CLI keywords, extracting the one or more CLI keywords from the input command, and arranging the one or more CLI keywords into the CLI command according to the CLI syntax of the input command;

translate an output message from a CLI format into XML format having the CLI syntax, wherein the output message is generated in the CLI format by the routing system responsive to the performance of the operation, and wherein the translating includes parsing the output message to identify at least one CLI token, translating each CLI token of the output message into a corresponding XML value according to a stored mapping of CLI tokens-to-XML values, and generating the output message in the XML format with the XML values; and transmit the output message in the XML format having the CLI syntax to a remote device external from the routing system.

7. The computer-usable memory device of claim **6**, wherein the input command is formatted in accordance with an XML schema of CLI rules and behaviors enforced by an internetwork operating system (IOS) command line interface (CLI) parser subsystem.

8. The computer-usable memory device of claim **7**, wherein the translation of the input command from XML format having a CLI syntax into a CLI command comprises:

parsing the input command to identify an XML command attribute;

traversing the input after the XML command attribute to identify any keywords and any parameters associated with the XML command attribute;

translating the XML command attribute into the CLI command; and

translating the keywords and any parameters into associated attributes of the CLI command.

9. The computer-usable memory device of claim **6**, wherein the output message in the XML format having the CLI syntax includes data formatted in accordance with an XML data model of CLI rules and behaviors enforced by an internetwork operating system (IOS) command line interface (CLI) parser subsystem.

10. The computer-usable memory device of claim **6**, wherein the translation of the output message from the CLI format into the XML format having the CLI syntax comprises:

parsing the output message to identify at least one CLI token;

accessing a stored mapping of CLI tokens-to-XML values;

translating each CLI token of the output message into a corresponding XML value, in accordance with said stored mapping; and

generating the output message in the XML format having the CLI syntax with the XML values.

11. A system comprising:

an input port device to receive an input command requesting an operation be performed by a routing system, wherein the input command is configured in an extensible markup language (XML) format having a command line interface (CLI) syntax with CLI keywords sequenced according to configuration rules for CLI commands;

means for translating the input command from the XML format having the CLI syntax into a CLI command by identifying at least one XML tag that includes an XML parameter to indicate the XML tag includes one or more CLI keywords, extracting the one or more CLI keywords from the input command, and arranging the one or more CLI keywords into the CLI command according to the CLI syntax of the input command;

means for performing the operation responsive to the CLI command;

means for generating an output message in a CLI format responsive to the performance of the operation;

means for translating the output message from the CLI format into XML format having the CLI syntax, wherein the means for translating includes means for parsing the output message to identify at least one CLI token, means for translating each CLI token of the output message into a corresponding XML value according to a stored mapping of CLI tokens-to-XML values, and means for generating the output message in the XML format with the XML values; and

means for transmitting the output message in the XML format having the CLI syntax to a remote device external from the routing system.

12. The system of claim **11**, wherein the input command is formatted in accordance with an XML schema of CLI rules and behaviors enforced by an internetwork operating system (IOS) command line interface (CLI) parser subsystem.

13. The system of claim **12**, wherein the means for translating of the input command from XML format having a CLI syntax into a CLI command comprises:

means for parsing the input command to identify an XML command attribute;

means for traversing the input after the XML command attribute to identify any keywords and any parameters associated with the XML command attribute;

means for translating the XML command attribute into the CLI command; and

means for translating the keywords and any parameters into associated attributes of the CLI command.

14. The system of claim **11**, wherein the output message in the XML format having the CLI syntax includes data formatted in accordance with an XML data model of CLI rules and behaviors enforced by an internetwork operating system (IOS) command line interface (CLI) parser subsystem.

15. The system of claim **11**, wherein the means for translating of the output message from the CLI format into the XML format having the CLI syntax comprises:

means for parsing the output message to identify at least one CLI token;

means for accessing a stored mapping of CLI tokens-to-XML values;

means for translating each CLI token of the output message into a corresponding XML value, in accordance with said stored mapping; and

means for generating the output message in the XML format having the CLI syntax with the XML values.

* * * * *